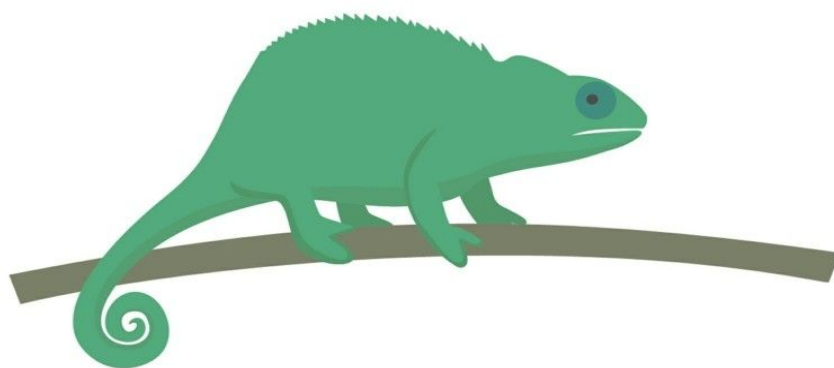


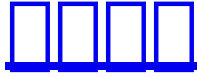
从基础概念到完整项目开发，  
帮助零基础读者快速掌握PyQt 5编程开发

# PyQt 5

## 快速开发与实战

王硕 孙洋洋 著





8Java3  
B/SJ2EEJSP/ServletJDBCStruts 2  
SpringHibernatePythonJavaScriptHTML 5  
Top





GUI

Top

PyQt 5

PyQt 5

PyQt 5

Publishing House of Electronics Industry

BEIJING

## 前言

本書介紹PyQt 5開發環境的搭建與PyQt 5開發環境的PyQt 5Qt  
Python開發環境的Qt開發環境的Python開發環境的Python  
Python開發環境的PyQt 5開發環境的PyQt 5開發環境的  
PyQt 5開發環境的PyQt 5開發環境的PyQt 5開發環境的  
Python開發環境的PandasMatplotlib  
PlotlyPyQt 5開發環境的PyQt 5開發環境的PyQt 5開發環境的  
PyQt 5開發環境

本書介紹PyQt 5開發環境的PyQt 5開發環境的  
Python開發環境的

本書介紹PyQt 5開發環境的PyQt 5開發環境的

本書介紹PyQt 5開發環境的PyQt 5開發環境的

本書介紹PyQt 5開發環境的PyQt 5開發環境的

PyQt 5開發環境/PyQt 5開發環境.—PyQt 5開發環境2017.10

ISBN 978-7-121-32291-4

I.①P... II.①...②... III.①PyQt—PyQt IV.①TP311.561

本書介紹PyQt 5開發環境的PyQt 5開發環境的2017176717

本書介紹PyQt 5開發環境的PyQt 5開發環境的

本書介紹PyQt 5開發環境的PyQt 5開發環境的

本書

本書

本書介紹PyQt 5開發環境的PyQt 5開發環境的173

本書100036

本書787×1092 1/16

本書35.75

本書756

本書2017101

2017101

99.00

0108825488888258888

zlbs@phei.com.cn  
dbqq@phei.com.cn

01051260888-819faq@phei.com.cn



5  
Tensorflow PyTorch Python

[illegible][illegible]

**Web**

Python Python Python Python  
10 CCIE

20 IT  
 VBS C# Python  
 JavaScript ISTQB

10 Java Python Web  
Drupal8 CMS

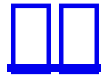
□□□□□□□□18□□□□□10□□□□□□□□□□□□□□□□□□□  
□□□□□□□□□□□□□2013—2015□□□10□□□□□

3C++UE4VS Cocos2dxC/C++Python

OracleTalkingData

FRM Level ICTA

10



Python 是一個簡單易學、強大且靈活的編程語言。Qt 是一個跨平台的 C++ 圖形用戶界面庫。Python 和 Qt 的組合可以通過 PyQt 實現。PyQt 是一個 Python 和 Qt 的橋樑，它允許 Python 程序使用 Qt 的圖形用戶界面庫。PyQt 有兩個版本：PyQt4 和 PyQt5。PyQt4 是基於 Qt4 的，而 PyQt5 是基於 Qt5 的。PyQt5 是 PyQt4 的後繼者，它提供了更多的功能和更好的性能。PyQt5 的安裝和使用相對簡單，並且有大量的文檔和社區支持。PyQt5 的代碼如下：

PyQt 是一個跨平台的 Python 圖形用戶界面庫。PyQt 0.1 是 PyQt 1 的前身。PyQt 2 是 PyQt 3 的前身。PyQt 4 是 PyQt 5.9 的前身。2017 年 8 月 9 日，PyQt 和 Qt 的開發者宣佈，PyQt 5 將成為 PyQt 4 的後繼者。PyQt 5 將基於 Qt 5，而 PyQt 4 將基於 Qt 4。PyQt 5 的代碼如下：

PyQt 5 是一個跨平台的 Python 圖形用戶界面庫。PyQt 5 的代碼如下：

本書は、PyQt 5のインストール、開発環境の構築、基本的な使い方、そして、PyQt 5の応用開発について、詳しく解説しています。PyQt 5のインストール、開発環境の構築、基本的な使い方、そして、PyQt 5の応用開発について、詳しく解説しています。

本書は、PyQt 5のインストール、開発環境の構築、基本的な使い方、そして、PyQt 5の応用開発について、詳しく解説しています。PyQt 5のインストール、開発環境の構築、基本的な使い方、そして、PyQt 5の応用開発について、詳しく解説しています。

本書は、PyQt 5のインストール、開発環境の構築、基本的な使い方、そして、PyQt 5の応用開発について、詳しく解説しています。PyQt 5のインストール、開発環境の構築、基本的な使い方、そして、PyQt 5の応用開発について、詳しく解説しています。

本書は

本書は11の章に分かれており、PyQtのインストール、開発環境の構築、基本的な使い方、そして、PyQtの応用開発について、詳しく解説しています。

1章はPyQtのインストール、開発環境の構築、基本的な使い方、そして、PyQtの応用開発について、詳しく解説しています。

2章はPythonのインストール、開発環境の構築、基本的な使い方、そして、Pythonの応用開発について、詳しく解説しています。

3章はQt Designerのインストール、開発環境の構築、基本的な使い方、そして、Qt Designerの応用開発について、詳しく解説しています。

4章はPyQt 5のインストール、開発環境の構築、基本的な使い方、そして、PyQt 5の応用開発について、詳しく解説しています。

5章はPyQt 5のインストール、開発環境の構築、基本的な使い方、そして、PyQt 5の応用開発について、詳しく解説しています。



第6章 PyQt 5 与 Qt Designer 3  
本章主要介绍 Qt Designer 的使用方法，包括如何创建新的工程、如何设计界面、如何保存工程文件等。本章还介绍了 Qt Designer 的一些常用工具，如属性浏览器、对象浏览器等。

第7章 PyQt 5 与 PyQt 5 的集成  
本章主要介绍 PyQt 5 与 PyQt 5 的集成方法，包括如何安装 PyQt 5、如何配置环境变量等。

第8章 PyQt 5 与 PyQt 5 的集成

第9章 PyQt 5 与 Python 的集成  
本章主要介绍 PyQt 5 与 Python 的集成方法，包括如何安装 PyQt 5、如何配置环境变量等。本章还介绍了 PyQt 5 的一些常用库，如 PyInstaller、Pandas、Matplotlib、PyQtGraph、Plotly 等。

第10章 PyQt 5 与 PyQt 5 的集成

第11章 PyQt 5 与 PyQt 5 的集成

附录 A

附录 B

附录 C

附录 D

附录 E

附录 F

附录 G

附录 H

[illegible][illegible][illegible][illegible][illegible][illegible][illegible]

IOS Dream Python  
Android Hank

[illegible]

Rocky English  
Rocky English  
Rocky English

“ ”

100%

[illegible]

PC

[www.broadview.com.cn](http://www.broadview.com.cn)

● 本公司 代理 各种 品牌 的 汽车 保险 业务 代理 各种 品牌 的 汽车 保险 业务

● 本公司 代理 各种 品牌 的 汽车 保险 业务 代理 各种 品牌 的 汽车 保险 业务

本公司 代理 各种 品牌 的 汽车 保险 业务 代理 各种 品牌 的 汽车 保险 业务

● 本公司 代理 各种 品牌 的 汽车 保险 业务 代理 各种 品牌 的 汽车 保险 业务

本公司 代理 各种 品牌 的 汽车 保险 业务 代理 各种 品牌 的 汽车 保险 业务  
<http://www.broadview.com.cn/32291>





[□□](#)

[□□□□](#)

[□□](#)

[□□□□](#)

[□□□□□□](#)

[□□](#)

[□1□□□PyQt 5](#)

[1.1 PyQt□□□□](#)

[1.1.1 PyQt 5□□□](#)

[1.1.2 Qt□PyQt□□□](#)

[1.1.3 □□□□□□□□□□](#)

[1.1.4 PyQt 4/PyQt 5](#)

[1.1.5 Python 2/Python 3](#)

[1.2 PyQt 5□□□□](#)

[1.2.1 □Windows□□□PyQt 5□□](#)

[1.2.2 □Mac OS□□□PyQt 5□□](#)

[1.2.3 PyQt 5□□□□□](#)

[1.2.4 □□□□Python□□](#)

[1.2.5 □□PyQt 5□API□□](#)

[1.3 Eric 6□□□□□□](#)

[1.3.1 Eric 6□□□□□□](#)

[1.3.2 Eric 6□□□□□](#)

[1.3.3 □□□□□□□jedi](#)

[1.3.4 □□Eric 6](#)

[1.3.5 Eric 6□□□□□](#)

[1.4 □□□□□□](#)

## [02 Python 入門](#)

### [2.1 Python のインストール](#)

### [2.2 変数](#)

#### [02-1 変数の宣言](#)

### [2.3 String \(文字列\)](#)

#### [02-2 文字列の宣言](#)

#### [02-3 文字列の操作](#)

### [2.4 List \(リスト\)](#)

#### [02-4 リストの宣言](#)

### [2.5 Tuple \(タプル\)](#)

#### [02-5 タプルの宣言](#)

### [2.6 Dictionary \(辞書\)](#)

#### [02-6 辞書の宣言](#)

### [2.7 条件分岐](#)

### [2.8 繰り返し](#)

#### [02-7 for 文](#)

### [2.9 関数](#)

#### [02-8 関数の宣言](#)

### [2.10 partial](#)

#### [02-9 partial 関数](#)

### [2.11 lambda](#)

#### [02-10 lambda 関数](#)

### [2.12 例外](#)

#### [02-11 try-except](#)

### [2.13 例外](#)

#### [02-12 try-except-else](#)

### [2.14 例外](#)

#### [02-13 try-except-finally](#)

### 3 Qt Designer

#### 3.1 Qt Designer

##### 3.1.1

##### 3.1.2

##### 3.1.3

##### 3.1.4

##### 3.1.5

#### 3.2

##### 3.2.1

##### 3.2.2

#### 3.3 Qt Designer

##### 3.3.1

##### 3.3.2

##### 3.3.3

##### 3.3.4

#### 3.4

##### 3.4.1

##### 3.4.2

#### 3.5

##### 3.5.1

##### 3.5.2

##### 3.5.3

#### 3.6

##### 3.6.1

##### 3.6.2

##### 3.6.3

##### 3.6.4

### 4 PyQt 5

## 4.1 QMainWindow

### 4.1.1

### 4.1.2

#### 4-1

### 4.1.3

#### 4-2

### 4.1.4

#### 4-3

## 4.2 QWidget

### 4.2.1

### 4.2.2

#### 4-4

### 4.2.3 PyQt 5

#### 4-5

### 4.2.4

#### 4-6

### 4.2.5

## 4.3 QLabel

### 4-7 QLabel

### 4-8 QLabel

## 4.4

### 4.4.1 QLineEdit

#### 4-9 EchoMode

#### 4-10

#### 4-11

#### 4-12

### 4.4.2 QTextEdit

#### 4-13 QTextEdit



## 4.5 [□□□□□](#)

### 4.5.1 [QAbstractButton](#)

### 4.5.2 [QPushButton](#)

[□□4-14 \[QPushButton\]\(#\)□□□□□](#)

### 4.5.3 [QRadioButton](#)

[□□4-15 \[QRadioButton\]\(#\)□□□□□](#)

### 4.5.4 [QCheckBox](#)

[□□4-16 \[QCheckBox\]\(#\)□□□□□](#)

## 4.6 [QComboBox](#)□□□□□□□

[□□4-17 \[QComboBox\]\(#\)□□□□□](#)

## 4.7 [QSpinBox](#)□□□□□

[□□4-18 \[QSpinBox\]\(#\)□□□](#)

## 4.8 [QSlider](#)□□□□□

[□□4-19 \[QSlider\]\(#\)□□□](#)

## 4.9 [□□□□□□□](#)

### 4.9.1 [QDialog](#)

[□□4-20 \[QDialog\]\(#\)□□□](#)

### 4.9.2 [QMessageBox](#)

[□□4-21 \[QMessageBox\]\(#\)□□□](#)

### 4.9.3 [QInputDialog](#)

[□□4-22 \[QInputDialog\]\(#\)□□□](#)

### 4.9.4 [QFontDialog](#)

[□□4-23 \[QFontDialog\]\(#\)□□□](#)

### 4.9.5 [QFileDialog](#)

[□□4-24 \[QFileDialog\]\(#\)□□□](#)

## 4.10 [□□□□□□□□](#)

### 4.10.1 [QPainter](#)

[□□4-25 \[□□□□\]\(#\)](#)

[□□4-26 □□□](#)

#### [4.10.2 QPen](#)

[□□4-27 QPen□□□](#)

#### [4.10.3 QBrush](#)

[□□4-28 QBrush□□□](#)

#### [4.10.4 QPixmap](#)

[□□4-29 QPixmap□□□](#)

### [4.11 □□□□□□](#)

#### [4.11.1 Drag□Drop](#)

[□□4-30 □□□□](#)

#### [4.11.2 QClipboard](#)

[□□4-31 QClipboard□□□](#)

### [4.12 □□□□□□](#)

#### [4.12.1 QCalendar](#)

[□□4-32 QCalendar□□□](#)

#### [4.12.2 QDateTimeEdit](#)

[□□4-33 QDateTimeEdit□□□](#)

### [4.13 □□□□□□□□□□](#)

#### [4.13.1 □□□](#)

[□□4-34 QMenuBar□□□](#)

#### [4.13.2 QToolBar](#)

[□□4-35 QToolBar□□□](#)

#### [4.13.3 QStatusBar](#)

[□□4-36 QStatusBar□□□](#)

### [4.14 QPrinter](#)

[□□4-37 QPrinter□□□](#)

## [□5□ PyQt 5□□□□□□□](#)

### [5.1 □□□□](#)

### 5.1.1 QTableView

## 5-1 QTableView

### 5.1.2 QListView

## 5-2 QListView

### 5.1.3 QListWidget

## 5-3 QListWidget

### 5.1.4 QTableWidget

### 5.1.5 QTreeView

## 5.2 □□□□□□□□

### 5.2.1 QTabWidget

## 5-4 QWidget

### 5.2.2 QStackedWidget

## 5-5 QStackedWidget

### 5.2.3 QDockWidget

## 5-6 QDockWidget

### 5.2.4 同値関係

□□5-7 □□□□□□

### 5.2.5 QScrollBar

## □□5-8 QScrollBar

### 5.3 □□□

### 5.3.1 QTimer

### 5.3.2 QThread

□□5-9 □□□□□□□□UI□□□□□□□□

### 5.3.3 同位素

## 5.4 □□□□

□□5-10 □□□□□□□□Web□□

□□5-11 □□□□□□□□Web□□

5-12 用HTML

[5-13 PyQt와 JavaScript](#)

[5-14 JavaScript와 PyQt](#)

## [6 PyQt 5](#)

[6.1](#)

[6.2 PyQt 5](#)

[6.3 PyQt 5](#)

[6.4 QVBoxLayout](#)

[6.4.1 QHBoxLayout](#)

[6.4.2 QVBoxLayout](#)

[6.4.3 addStretch\(\)](#)

[6.5 QGridLayout](#)

[6.5.1](#)

[6.5.2](#)

[6.6 QFormLayout](#)

[6.7](#)

[6.7.1](#)

[6.7.2](#)

[6.8 QSplitter](#)

## [7 PyQt 5](#)

[7.1](#)

[7.1.1](#)

[7.1.2](#)

[7.1.3](#)

[7.1.4](#)

[7.2](#)

[7.2.1](#)

[7.2.2](#)

[7.2.3](#)

#### 7.2.4 □□□□□□□□

### 7.3 □□□□□□□□

#### 7.3.1 □□□□□□□□

#### 7.3.2 □□□□□□

#### 7.3.3 □□□□□□

#### 7.3.4 □□□□□□□□

#### 7.3.5 Qt Designer□□□□□□□□□□□□

#### 7.3.6 □□□□□□□□□□

### 7.4 □□□□□□□

#### 7.4.1 □□□□□□□□□

#### 7.4.2 □□□□□□

#### 7.4.3 □□□□□□□□

#### 7.4.4 □□□□□□

### 7.5 □□□□□□

#### 7.5.1 □□□□□□□

#### 7.5.2 □□□□□□□□□□

#### 7.5.3 □□□□□□□□□□

## □8□ PyQt 5□□□□□

### 8.1 □□□□

#### 8.1.1 □□□□□□

#### □□8-1 □□□□□□

#### 8.1.2 □□□□□□

#### 8.1.3 □□□□□□□□□□

### 8.2 □□

#### 8.2.1 □□□

#### 8.2.2 □□□□

#### 8.2.3 □□□□□

#### □□8-2 □□□□□□□□□

## 図8-3 画面構成

### 8.3 QSSとUI

#### 8.3.1 QSS

#### 8.3.2 QSS

#### 8.3.3 QSS

#### 8.3.4 QSS

#### 8.3.5 QDarkStyleSheet

### 8.4 画面構成

#### 8.4.1 QSS

#### 8.4.2 QPalette

#### 8.4.3 paintEvent

### 8.5 画面構成

#### 8.5.1 画面構成

#### 8.5.2 GIF

### 8.6 画面構成

#### 8.6.1 画面構成

#### 8.6.2 画面構成

#### 8.6.3 画面構成

#### 8.6.4 画面構成

#### 8.6.5 QSS

## 9 PyQt 5

### 9.1 PyInstallerとEXE

### 9.2 画面構成

#### 9.2.1 SQLite

#### 9.2.2 画面構成

#### 9.2.3 SQL

#### 9.2.4 画面構成

#### 9.2.5 画面構成

## [9.3 PandasPyQt](#)

### [9.3.1 qtpandas](#)

### [9.3.2](#)

### [9.3.3](#)

### [9.3.4 qtpandas](#)

## [9.4 MatplotlibPyQt](#)

### [9.4.1 MatplotlibWidget](#)

### [9.4.2](#)

### [9.4.3 MatplotlibWidget](#)

### [9.4.4](#)

## [9.5 PyQtGraphPyQt](#)

### [9.5.1 PyQtGraph](#)

### [9.5.2](#)

### [9.5.3](#)

### [9.5.4 PyQtGraph](#)

### [9.5.5](#)

## [9.6 PlotlyPyQt](#)

### [9.6.1 Plotly](#)

### [9.6.2](#)

### [9.6.3](#)

### [9.6.4 Plotly\\_PyQt5](#)

### [9.6.5](#)

### [9.6.6 PlotlyPyQt 5.6](#)

### [9.6.7](#)

## [9.7 UI](#)

### [9.7.1](#)

### [9.7.2](#)

### [9.7.2 Python](#)

[9.7.3](#) [□□□□□□](#)

[9.7.4](#) [□□□□□□](#)

[9.7.5](#) [□□□□□□](#)

[□10□ PyQt 5□□□□□□□□□□](#)

[10.1](#) [□□□□□□□□](#)

[10.1.1](#) [□□□□□□](#)

[10.1.2](#) [□□□□□□□□□□□□□□API](#)

[10.1.3](#) [□□□□](#)

[10.1.4](#) [□□□□□□□□□□.py□□](#)

[10.1.5](#) [□□□□□□□□](#)

[10.2](#) [□□□□](#)

[10.2.1](#) [□□□□□□□□](#)

[10.2.2](#) [□□□□](#)

[10.3](#) [□□□□□□□□](#)

[□11□ PyQt 5□□□□□□□□□□](#)

[11.1](#) [□□□□□□□□□□](#)

[11.2](#) [□□□□□□□□□□](#)

[11.3 PyQt 5□□□□□□□□□□□□](#)

[11.3.1](#) [□□□□□□□□□□](#)

[11.3.2](#) [□□□□□□□□□□](#)

[11.4 PyQt 5□□□□□□□□□□](#)

[11.5 PyQt 5□□□□□□□□](#)

[11.5.1](#) [□□□□□□](#)

[11.5.2](#) [□□□□](#)

[□□□□](#)



# 1 PyQt 5

## 1.1 PyQt

GUI GUI

PyQt GUI

GUI

GUI Graphical User Interface GUI

GUI GUI

Python GUI C/C++ GUI Toolkit Python

Python GUI PyQt Tkinter wxPython Kivy PyGUI Libavg PyQt Qt Python GUI

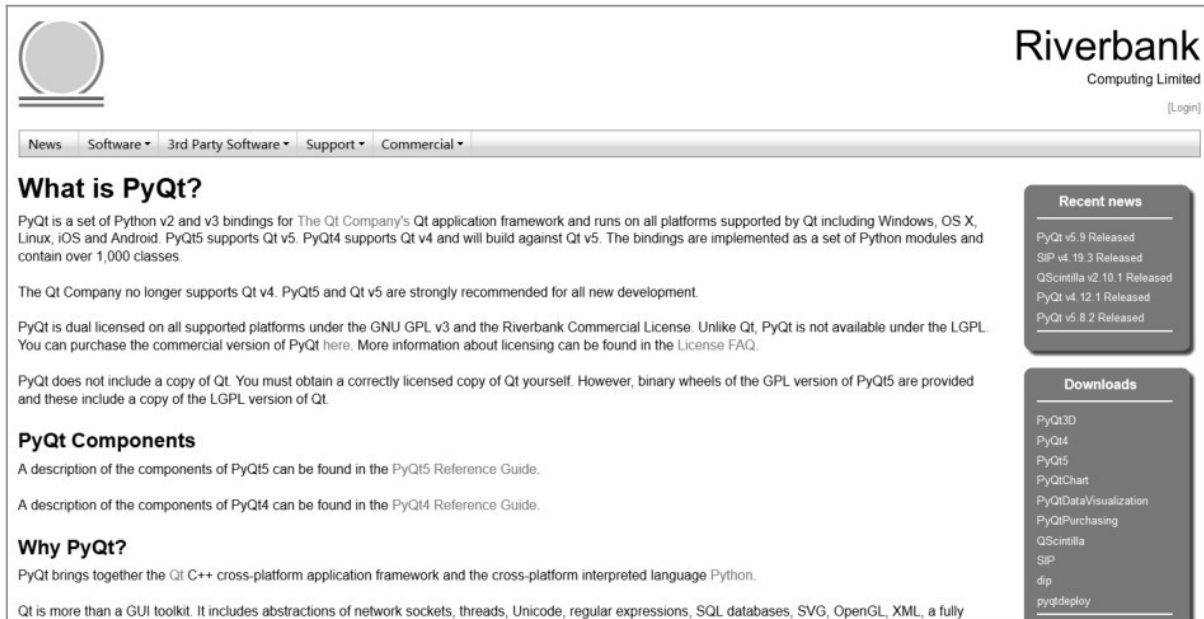
PyQt

PyQt GUI Python Qt GUI

PyQt Phil Thompson Python PyQt 620 6000 PyQt UNIX Windows

Mac OS PyQt GPL GPL  
UNIX PyQt4 GPL

PyQt 5 <https://www.riverbankcomputing.com/>  
1-1 PyQt 5.9 PyQt Python  
Python Qt



1-1

Qt Trolltech C++ GUI  
IDE GUI Qt  
Python Qt  
Qt Windows Linux Mac OS

2008 6 Trolltech Nokia Qt  
2012 8 Qt IT Digia Qt  
Digia LGPL GPL Qt

PyQt 5 的 GPL 许可证与 GNU General Public License 兼容。PyQt 5 的 GPL 许可证允许您在商业软件中使用 PyQt 5。

GPL 许可证要求您必须遵守 GPL 许可证的所有条款。如果您使用 GPL 许可证的代码，您必须将您的代码也发布为 GPL 许可证。GPL 许可证要求您必须遵守 GPL 许可证的所有条款。

PyQt 5 的 GPL 许可证与 PyQt 4 的 GPL 许可证兼容。PyQt 5 的 GPL 许可证允许您在商业软件中使用 PyQt 5。

### 1.1.1 PyQt 5 许可证

Qt 是一个跨平台的 C++ 库，PyQt 是一个 Python 库。PyQt 3、PyQt 4 和 PyQt 5 都是 PyQt 的分支。PyQt 3 是在 1998 年发布的，PyQt 4 是在 2006 年发布的，PyQt 5 是在 2012 年发布的。PyQt 5 的 GPL 许可证允许您在商业软件中使用 PyQt 5。

PyQt 5 的 GPL 许可证允许您在商业软件中使用 PyQt 5。

PyQt 5 的 GPL 许可证要求您必须遵守 GPL 许可证的所有条款。如果您使用 GPL 许可证的代码，您必须将您的代码也发布为 GPL 许可证。PyQt 5 的 GPL 许可证允许您在商业软件中使用 PyQt 5。

PyQt 5 的 GPL 许可证允许您在商业软件中使用 PyQt 5。PyQt 5 的 GPL 许可证允许您在商业软件中使用 PyQt 5。



图1-2

《财富》全球500强企业中的前10家企业，有8家使用Qt



图1-3

PyQt是Python语言与Qt的结合，PyQt是Python语言与Qt的结合

- 使用Qt的GUI库
- 支持Windows、Linux、Mac OS等操作系统
- 支持signal/slot信号槽机制
- 支持Qt的线程库
- 支持Qt的IDE Qt Designer

Python语言

- 支持多线程编程

## 1.1.2 Qt与PyQt

PyQtQtPythonPyQtQtQtAPIPyQtqmakeQ\_OBJECTPyQtPyQt

PyQtQtC++QtPythonPythonC++QtPyQtPython50%60%PythonC++Python

PyQtPythonQtPythonQtQtPyQtQtQt/callbackQt/PyQt

### 1.1.3

PythonGUIPythonPythonPythonGUIPythonGUI

#### **1.Tkinter**

TkinterPythonTk GUIPythonTclPythonTclTkinterTclTclPythonGUITkPerlTkTkC

TkinterPythonGUIPythonTk GUIPython WindowsIDLETkinterGUI

#### **2.wxPython**

wxPython Python GUI wxWidgets C++  
Python

wxPython Tkinter

### 3. PyGTK

PyGTK Python GTK+ GUI

PyGTK Tkinter Gnome GUI  
PyGTK BitTorrent GIMP

PyGTK Gedit Windows  
GTK GUI

### 4. PySide

PySide Qt 1.2.4 Qt 4.8  
PySide Python GUI Qt Python

PySide Tkinter LGPL 2.1

Qt Designer UI  
UI Python  
PySide Qt Python  
Qt Qt 4.8  
2015 10 14 Python PyQt GUI  
PyQt

## 1.1.4 PyQt 4/PyQt 5

PyQt 5 PyQt 4 PyQt 5  
1 PyQt 5 Python 2.6 Python 3  
Python 3 Python 2.7 PyQt 5



5 PyQt 5 Python 3 Python 2

Python 3 Python 3

## 1.2 PyQt 5

PC PyQt 5 Python 3

PyQt 5 Python github  
<https://github.com/cxinping/PyQt5>

### 1.2.1 Windows PyQt 5

Windows PyQt 5 1-1

1-1

操作系统	Windows 8 64 位平台
Python	3.5.3
PyQt	5.9
Eric	6.17

1-4

Python Python Python





□1-4

Python 2.7 Python 3.5  
Python 2 Python 3  
Python 3 Python 2  
Python 2 Python 3  
Python 2 Python 3

`Python 3`

## 1. Python 3

Python <https://www.python.org>

PyQt 5.9 Python 3 Python  
 3.5 Python 3.5.3  
<https://www.python.org/downloads/windows/> PyQt 5  
 Python 3.5.3 Windows  
 32 x86 64 x86-64  
 .exe 1-5



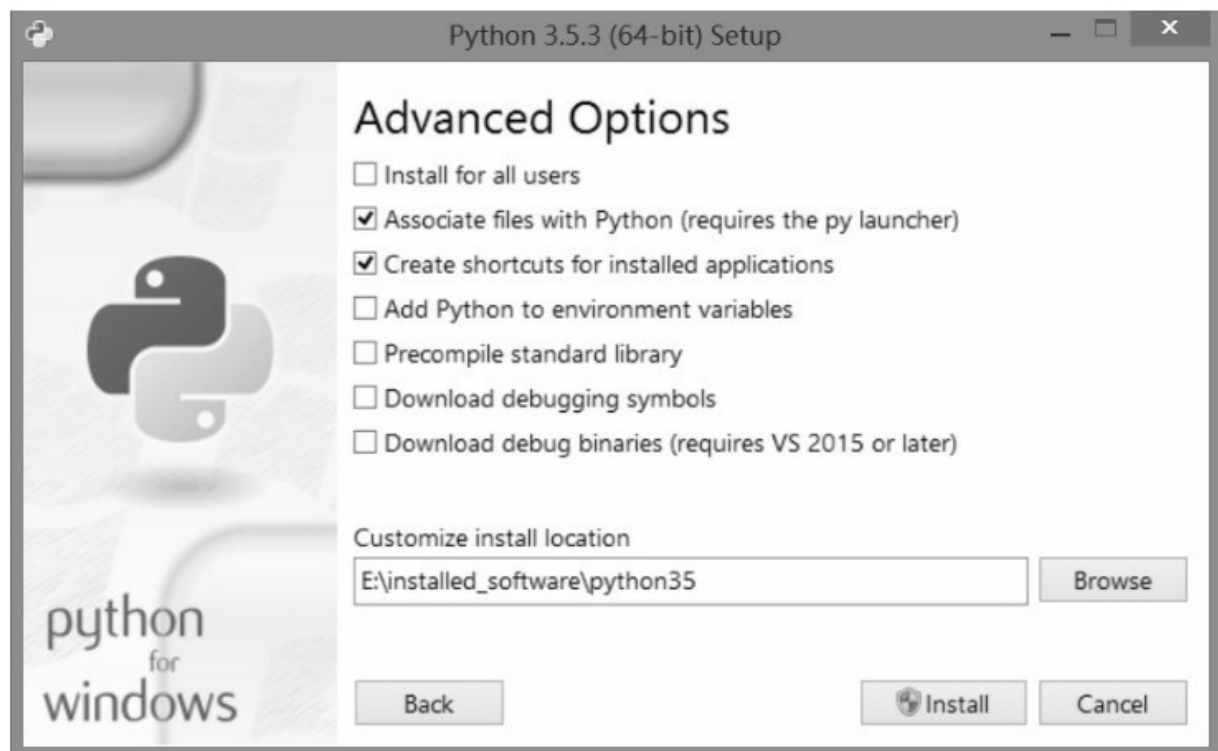
□1-5

```
Python 3.5.31-6
E:\installed_software\python351-7

```



□1-6



□1-7

如何安装Python 3.5 1-8

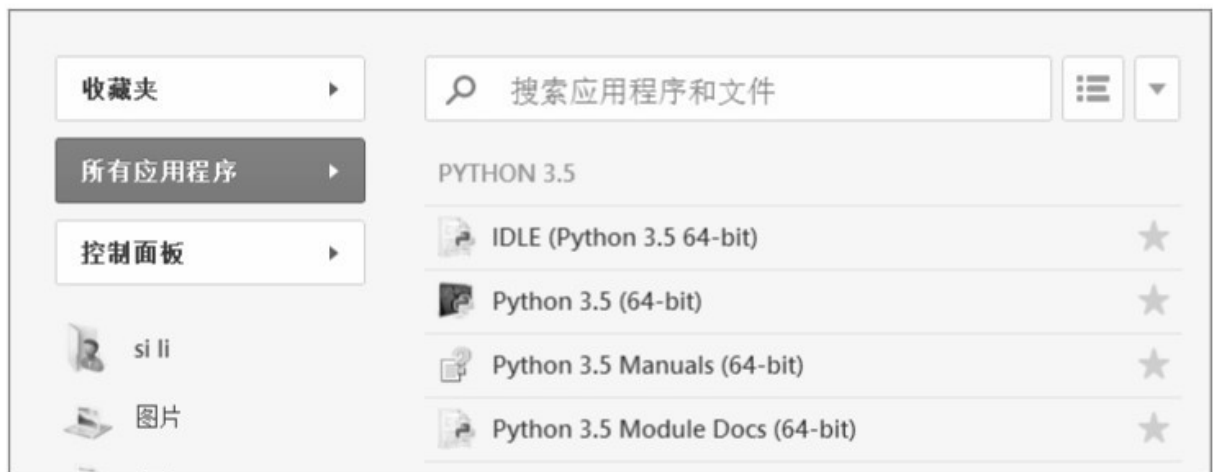


图1-8

如何安装Python 3.5 IDLE(Python 3.5 64-bit) 1-9  
Python Shell 1-9

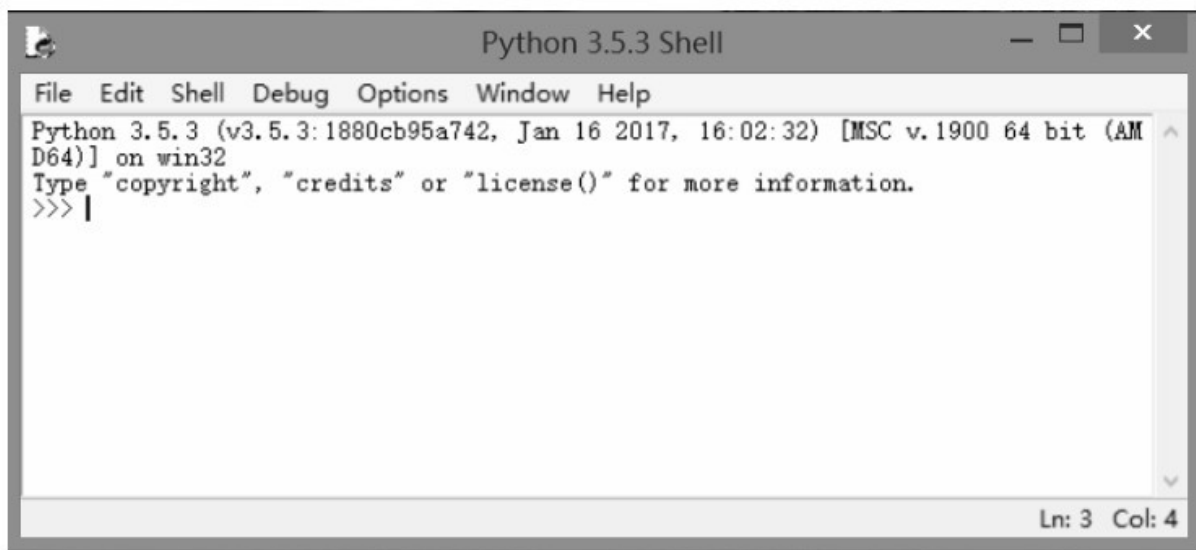


图1-9

如何安装Python 3.5 Path 1-10



图1-10

Path

E:/installed\_software/python35;E:/installed\_software/python35/Scripts;

E:/installed\_software/Python35 Python 3.5.3

1-11



图1-11

在Windows中安装Python后，可以通过“Win+R”打开cmd窗口，运行DOS命令，如图1-12所示。

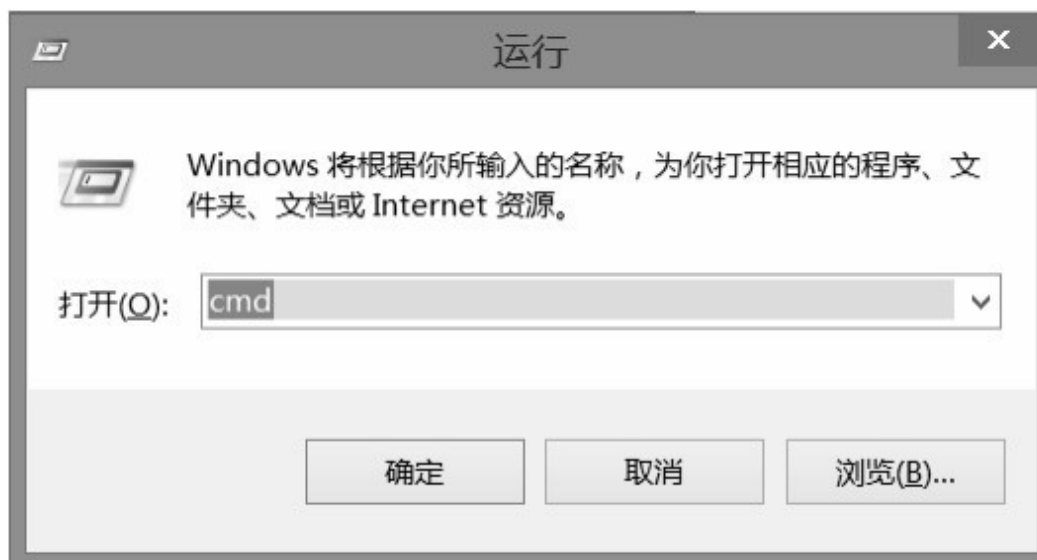


图1-12

下面将使用python解释器Python IDLE Python解释器来运行Python程序。图1-13展示了Python解释器的运行界面。

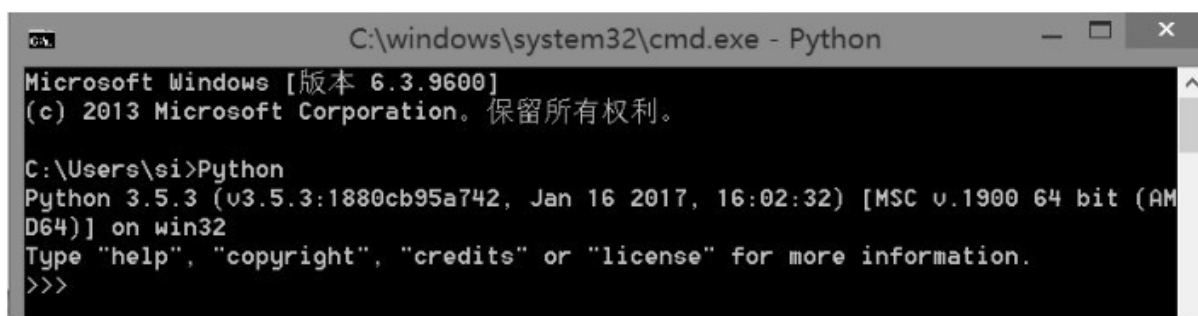


图1-13

下面将使用Python 3解释器来运行Python程序。图1-14展示了Python 3解释器的运行界面。

```
print("hello python")
```

下面将使用Python解释器来运行Python程序。图1-14展示了Python解释器的运行界面。

下面将使用Python 3解释器来运行Python程序。图1-15展示了Python 3解释器的运行界面。





```
>>> help("print")
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file: a file-like object (stream); defaults to the current sys.stdout.
    sep: string inserted between values, default a space.
    end: string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

01-17

按“Ctrl+C”按退出(exit())

## 2. PyQt 5

PyQt 5 下载地址 <https://www.riverbankcomputing.com/> 下载  
 PyQt 5 版本 5.9 下载 PyQt 5 版本 5.9 下载 PyQt 5 版本 5.9 下载  
 pip install PyQt5 安装 PyQt 5 版本 5.9 下载  
 pip install PyQt5 安装 PyQt 5 版本 5.9 下载  
 Python 版本 3.6 下载 Python 版本 3.6 下载 Python 版本 3.6 下载  
 Python 版本 3.6 下载 Python 版本 3.6 下载 Python 版本 3.6 下载  
<https://pypi.douban.com/simple> 安装 PyQt 5 版本 5.9 下载

pip install PyQt5-i <https://pypi.douban.com/simple>

01-18

```

99% | 77.1MB 209kB/s eta 0:00:0
99% | 77.1MB 148kB/s eta 0:00:0
99% | 77.1MB 172kB/s eta 0:00:0
99% | 77.1MB 173kB/s eta 0:00:0
99% | 77.1MB 173kB/s eta 0:00:0
99% | 77.1MB 173kB/s eta 0:00:0
99% | 77.1MB 173kB/s eta 0:00:0
99% | 77.1MB 173kB/s eta 0:00:0
99% | 77.2MB 174kB/s eta 0:00:0
99% | 77.2MB 173kB/s eta 0:00:0
99% | 77.2MB 173kB/s eta 0:00:0
99% | 77.2MB 192kB/s eta 0:00:0
99% | 77.2MB 212kB/s eta 0:00:0
99% | 77.2MB 162kB/s eta 0:00:0
99% | 77.2MB 163kB/s eta 0:00:0
100% | 77.2MB 213kB/s
Requirement already satisfied: sip<4.20,>=4.19.3 in e:\installed_software\python35\lib\site-packages
(from PyQt5)
Installing collected packages: PyQt5
Successfully installed PyQt5-5.9

```

01-18

PyQt 5.9 安装 Qt Designer 安装  
Liguist 安装 Qt

pip install PyQt5-tools-i <https://pypi.douban.com/simple>

01-19

```

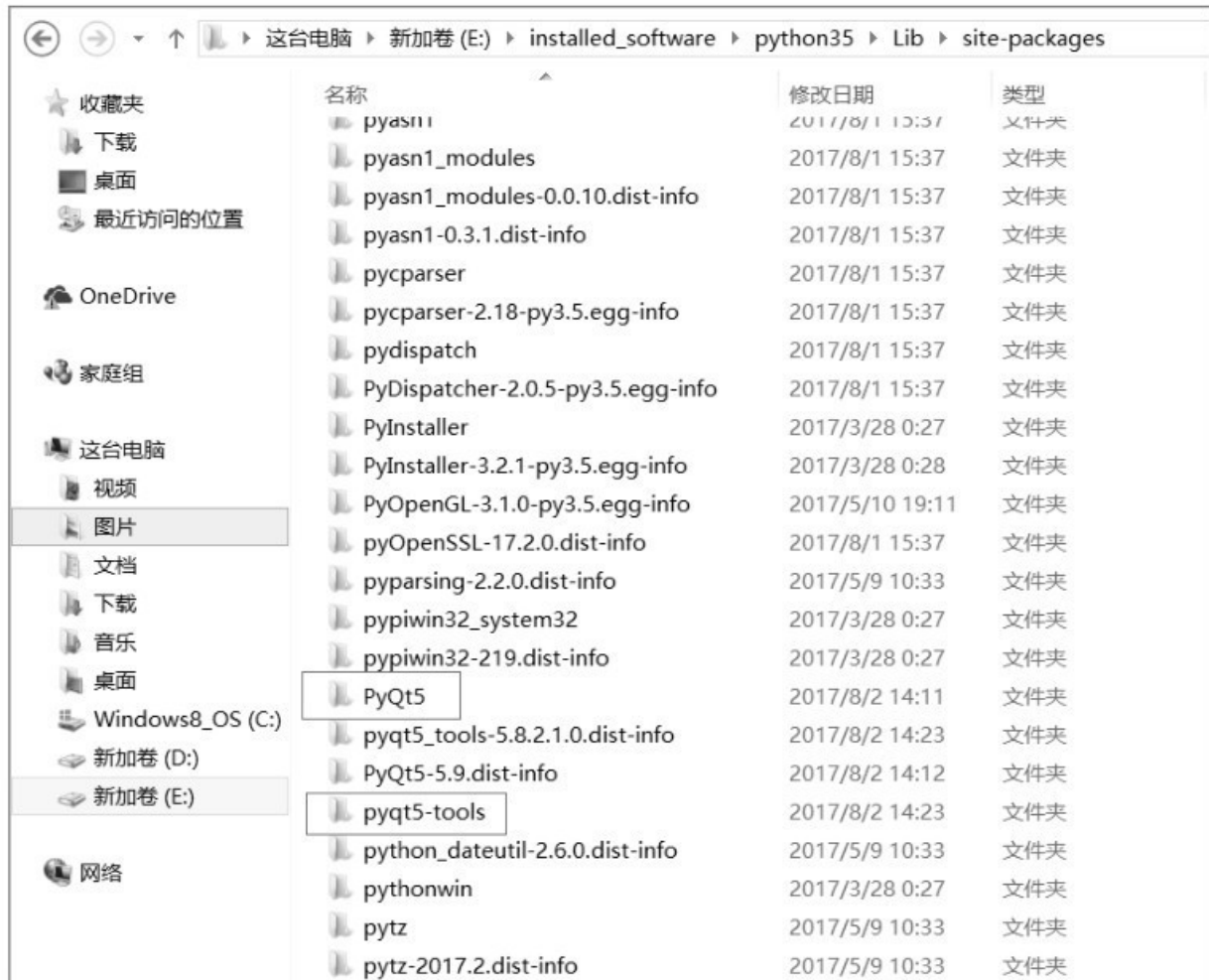
99% | 36.0MB 193kB/s eta 0:00:0
99% | 36.0MB 177kB/s eta 0:00:0
99% | 36.0MB 174kB/s eta 0:00:0
99% | 36.1MB 174kB/s eta 0:00:0
99% | 36.1MB 173kB/s eta 0:00:0
99% | 36.1MB 175kB/s eta 0:00:0
99% | 36.1MB 174kB/s eta 0:00:0
99% | 36.1MB 234kB/s eta 0:00:0
99% | 36.1MB 175kB/s eta 0:00:0
99% | 36.1MB 175kB/s eta 0:00:0
99% | 36.1MB 148kB/s eta 0:00:0
99% | 36.1MB 159kB/s eta 0:00:0
99% | 36.1MB 207kB/s eta 0:00:0
100% | 36.2MB 207kB/s
Installing collected packages: PyQt5-tools
Successfully installed PyQt5-tools-5.8.2.1.0

```

01-19

pip install PyQt 5 PyQt5-tools  
%python35\Lib\site-packages PyQt5 pyqt5-

tools 000000000 E:\installed\_software\python35\Lib\site-packages0001-20000



01-20

000 Windows 000000000 PyQt5-tools 0000000000PyQt5-tools000000000000000Path0000000000“0000”0000000000“00”→“000000”→“00”000“0000”000000000Path000000000001-21000

E:/installed\_software/python35/Lib/site-packages/pyqt5-tools;

00

E:/installed\_software/python35/Lib/site-packages/pyqt5-tools  
PyQt5-tools



WindowsPATHPath  
python35PyQt5-tools1-22

```
C:\Users\si\Desktop>path
PATH= [E:/installed_software/python35/Lib/site-packages/pyqt5-tools;E:/installed_software/python35/Lib/site-packages/PyQt5;. ;E:/installed_software/python35;E:/installed_software/python35/Scripts;E:/installed_software/apache-maven-3.2.5\bin;. ;E:/installed_software/JDK1_7_1204/bin;C:\Program Files (x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCLS Client\;C:\windows\system32;C:\windows;C:\windows\System32\Wbem;C:\windows\System32\WindowsPowerShell\v1.0\;C:\Program Files\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files (x86)\ATI Technologies\ATI.ACE\Core-Static;C:\Program Files\Intel\WiFi\bin\;C:\Program Files\Common Files\Intel\WirelessCommon\;C:\Program Files (x86)\Common Files\lenovo\easyplussdk\bin;C:\ProgramData\Lenovo\ReadyApps;C:/Users/si/AppData/Roaming/npm;
```

1-22

PyQt 5  
PyQt5/Chapter01/qt101\_testPyQt.py.pyPython

```
import sys
from PyQt5 import QtWidgets,QtCore
app=QtWidgets.QApplication(sys.argv)
widget=QtWidgets.QWidget()
widget.resize(360,360)
widget.setWindowTitle("hello,pyqt5")
widget.show()
sys.exit(app.exec_())
```

Windowsqt101\_testPyQt.pyWindows

```
python qt101_testPyQt.py
```

1-23WidgetPyQt 5

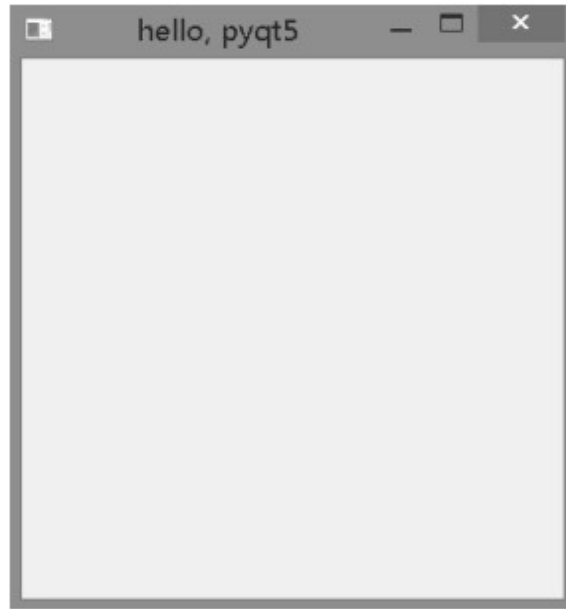


图1-23

## 1.2.2 在Mac OS上安装PyQt 5

本节介绍在Mac OS上安装PyQt 5的方法。PyQt 5依赖于Python 3.5，因此在安装PyQt 5之前，需要先安装Python 3.5。Mac OS上安装PyQt 5的方法与Linux和Windows系统类似，但有一些细节需要注意。

在Mac OS上安装PyQt 5时，需要安装以下依赖项：SIP、Qt和Python。本节将介绍如何安装这些依赖项，并给出安装PyQt 5的命令。

表1-2

操作系统	Mac OS X El Capitan 版本 10.11.5
Python	3.5.3
Qt	qt-opensource-mac-x64-5.9.1
PyQt	5.9
SIP	4.19.3

本节将介绍如何在Mac OS上安装PyQt 5.9。安装过程分为以下几个步骤：安装Python 3.5、安装Qt、安装SIP和安装PyQt 5.9。

名称	修改日期	大小	种类
 PyQt5_gpl-5.9.tar.gz	今天 下午4:26	3.1 MB	gzip 压缩归档
 python-3.5.3-macosx10.6.pkg	2017年1月25日 上午11:48	24.8 MB	安装器软件包
 qt-opensource-mac-x64-5.9.1.dmg	今天 下午5:18	3.74 GB	磁盘映像
 sip-4.19.3.tar.gz	今天 下午4:47	1 MB	gzip 压缩归档

图1-24

## 1. 安装Qt 5.9.1

将app文件Qt 5.9.1解压到1-25文件夹中，然后将文件夹移动到/Users/xinping/Qt5.9.1文件夹中，如图1-26所示。xinping为Mac OS用户名，请根据实际情况替换。



图1-25



图1-26

图1-27





图 1-27

图 1-27 所示为 Qt 5.9.1 安装时的组件选择界面。图 1-28 所示为 Qt 5.9.1 安装时的“安装”按钮。



图1-28

勾选“Launch Qt Creator”选项，然后点击“完成”按钮，即可启动Qt Creator。Qt Creator启动后，会显示Qt Creator的欢迎界面，如图1-29所示。

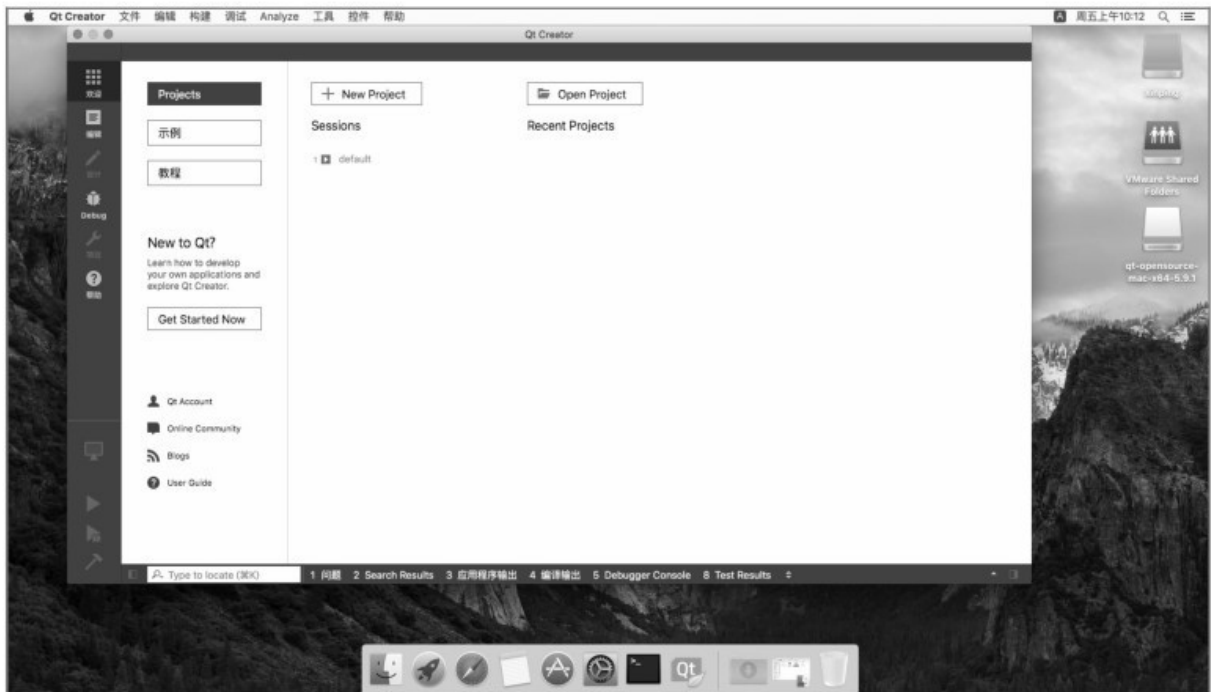


图1-29

## 2. 安装Python 3.5.3

访问<https://www.python.org/downloads/mac-osx/> Mac 安装Python 3.5.3的.pkg包 python-3.5.3-macosx10.6.pkg  
 安装pkg包 python-3.5.3-macosx10.6.pkg

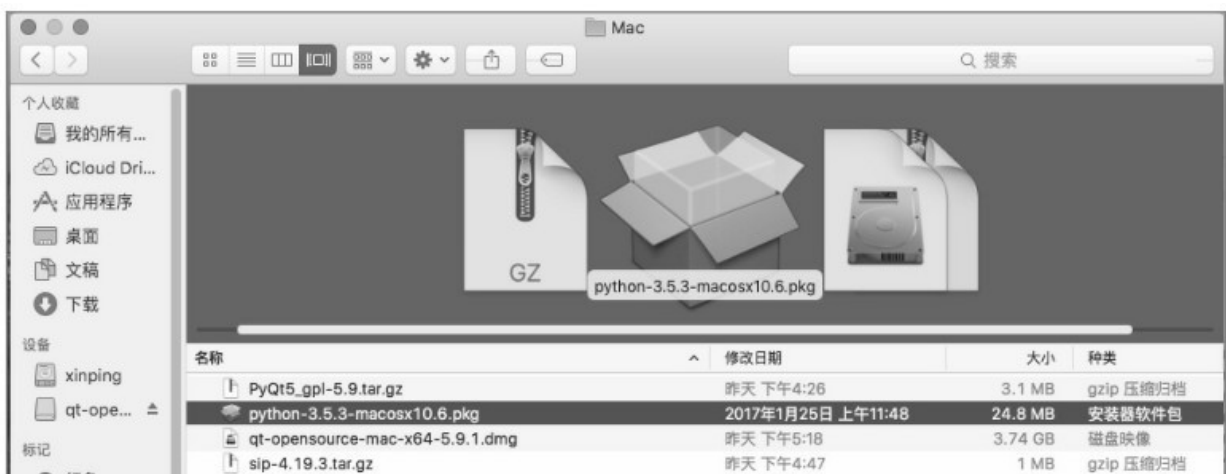


图1-30

□□□□□1-31□□1-32□□1-33□□□



□1-31





图1-33

### 3. SIP 4.19.3

```

$ curl https://www.riverbankcomputing.com/software/sip/download
$ tar xvf sip-4.19.3.tar.gz
$ cd sip-4.19.3
$ python3.5 configure.py --prefix=/Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/site-packages
$ make
$ make install
```

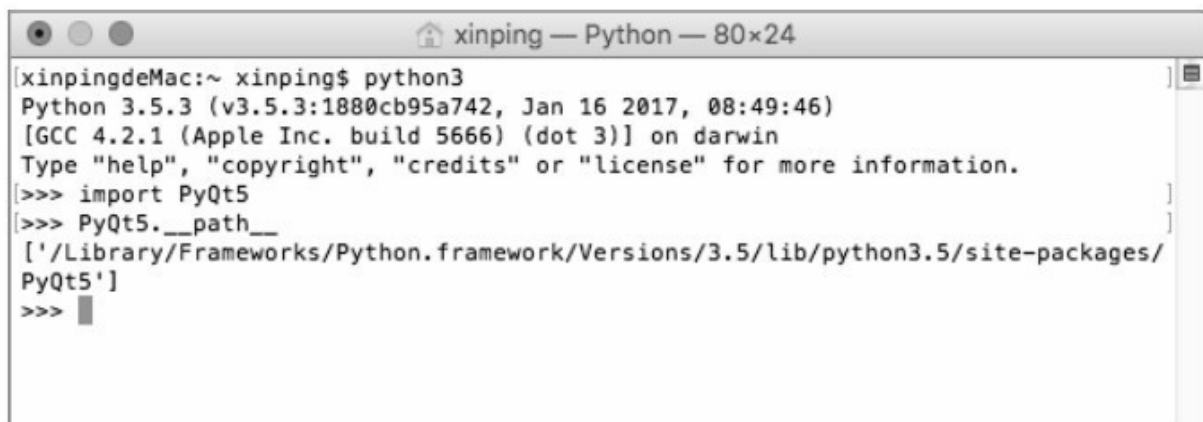
## 4. 安装 PyQt 5.9

```

# 从 Riverbank Computing 网站下载 PyQt 5.9 的源代码
https://riverbankcomputing.com/software/pyqt/download5
# 下载 PyQt-gpl-5.9.tar.gz 文件
tar xvf PyQt-gpl-5.9.tar.gz
cd PyQt-gpl-5.9
python3.5 configure.py --qmake
/Users/xinping/Qt5.9.1/5.9.1/clang_64/
bin/qmake -disable-QtPositioning-d
/Library/Frameworks/Python.framework/Versions/3.5/lib
/python3.5/site-packages
make
sudo make install
--qmake 指定 qmake 的路径为 "/Users/xinping/Qt5.9.1/5.9.1/clang_64/bin/qmake"
# 编译选项 -disable-QtPositioning 用于禁用 QtPositioning 模块
```

## 5. 验证安装

在 Terminal 中运行以下命令 1-34 验证安装是否成功



```
xinpingdeMac:~ xinping$ python3
Python 3.5.3 (v3.5.3:1880cb95a742, Jan 16 2017, 08:49:46)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import PyQt5
>>> PyQt5.__path__
['/Library/Frameworks/Python.framework/Versions/3.5/lib/python3.5/site-packages/PyQt5']
>>>
```

### 1.2.3 PyQt 5

在Windows中按下“Win+R”可以打开运行对话框，如图1-35所示。在“打开(O):”文本框中输入cmd，如图1-36所示。

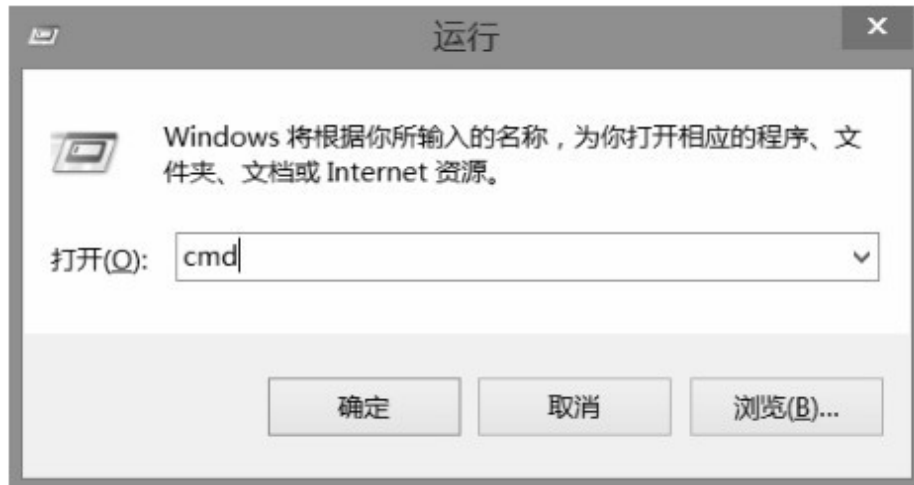


图1-35



图1-36

在Python 3.5.3中，可以使用以下命令安装PyQt 5。

```
import PyQt5
```

在Python 3.5.3中，可以使用以下命令安装PyQt 5。



help(PyQt5)

PyQt 5 1-37

```
C:\windows\system32\cmd.exe - python

PACKAGE CONTENTS
  QAxContainer
  Qsci
  Qt
  QtCore
  QtDesigner
  QtGui
  QtHelp
  QtLocation
  QtMultimedia
  QtMultimediaWidgets
  QtNetwork
  QtOpenGL
  QtPositioning
  QtPrintSupport
  QtQml
  QtQuick
  QtQuickWidgets
  QtSensors
  QtSerialPort
  QtSql
  QtSvg
  QtTest
  QtWebChannel
  QtWebEngineCore
  QtWebEngineWidgets
  QtWebSockets
  QtWidgets
  QtWinExtras
  QtXml
  QtXmlPatterns
  _QtOpenGLFunctions_2_0
  _QtOpenGLFunctions_2_1
  _QtOpenGLFunctions_4_1_Core
  uic (package)

FILE
e:\installed_software\python35\lib\site-packages\pyqt5\__init__.py
```

1-37

## 1.2.4 Python

Python 3.5 Python  
python35\Scripts pip easy\_install

pip easy\_install 1-38

这台电脑 > 新加卷 (E:) > installed_software > python35 > Scripts			
名称	修改日期	类型	大小
chrome_webbrowser.exe	2017/3/2 23:00	Windows 批处理...	1 KB
f2py.py	2017/5/9 10:33	PY 文件	1 KB
futurize.exe	2017/3/28 0:27	应用程序	73 KB
futurize-script.py	2017/3/28 0:27	PY 文件	1 KB
jsonschema.exe	2017/5/16 23:36	应用程序	96 KB
jupyter.exe	2017/5/16 23:36	应用程序	96 KB
jupyter-migrate.exe	2017/5/16 23:36	应用程序	96 KB
jupyter-troubleshoot.exe	2017/5/16 23:36	应用程序	96 KB
jupyter-trust.exe	2017/5/16 23:36	应用程序	96 KB
pasteurize.exe	2017/3/28 0:27	应用程序	73 KB
pasteurize-script.py	2017/3/28 0:27	PY 文件	1 KB
pip.exe	2017/3/22 17:43	应用程序	96 KB
pip3.5.exe	2017/3/22 17:43	应用程序	96 KB
pip3.exe	2017/3/22 17:43	应用程序	96 KB
pyi-archive_viewer.exe	2017/3/28 0:28	应用程序	73 KB
pyi-archive_viewer-script.py	2017/3/28 0:28	PY 文件	1 KB

1-38

matplotlib Python matplotlib  
setuptools numpy python-dateutil pytz pyparsing cyclery  
easy\_install pip

matplotlib

pip install matplotlib

matplotlib 1-39





```

C:\Users\si\Desktop>python
Python 3.5.3 (v3.5.3:1880cb95a742, Jan 16 2017, 16:02:32) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from PyQt5.QtWidgets import QWidget
>>> help( QWidget )
Help on class QWidget in module PyQt5.QtWidgets:

class QWidget(PyQt5.QtCore.QObject, PyQt5.QtGui.QPaintDevice)
|   QWidget(parent: QWidget = None, flags: Union[Qt.WindowFlags, Qt.WindowType] = Qt.WindowFlags())
|
|   Method resolution order:
|   QWidget
|   PyQt5.QtCore.QObject
|   sip.wrapper
|   PyQt5.QtGui.QPaintDevice
|   sip.simplewrapper
|   builtins.object
|
|   Methods defined here:
|
|   acceptDrops(...)
|       acceptDrops(self) -> bool
|
|   accessibleDescription(...)
|       accessibleDescription(self) -> str
|
|   accessibleName(...)
|       accessibleName(self) -> str
|
-- More --

```

1-41

help() PyQt 5

PyQt5/Chapter01/qt102\_PrintApi.py  
PyQt 5 QWidget

```

import sys
from PyQt5.QtWidgets import QWidget
out=sys.stdout
sys.stdout=open(r'E:\QWidget.txt' ,'w')
help( QWidget )
sys.stdout.close()
sys.stdout=out

```

E\QWidget.txt QWidget  
API Application Programming Interface 1-42





Eric Python IDE Python Qt GUI Eric IDE Eric 6 PyQt 5 Python GUI

- Windows/Linux/Mac OS
- 
- 
- 
- 
- 
- 
- if while
-

- 安装Python
- 安装Python解释器/库
- 安装Qt
- 安装Qt Designer/PyQt 5
- 安装PyQt 5
- 安装PyQt 5
- 安装PyQt 5

——Eric 6 安装指南  
 “Eric 6” 安装指南

### 1.3.1 Eric 6 安装指南

#### 1. 安装Eric 6

Eric 6 的官方网站是 <http://eric-ide.python-projects.org/>，  
<https://sourceforge.net/projects/eric-ide/files/eric6/stable/>  
 Windows 用户可以直接从 Eric 6 官方网站下载  
 Eric 6-17.04.1 版本



Looking for the latest version? **Download eric6-17.04.1.zip (20.8 MB)**

Home / eric6 / stable / 17.04.1 RSS

Name	Modified	Size	Downloads / Week
↑ Parent folder			
changelog	<a href="#">2017-04-09</a>	13.5 kB	4 <input type="checkbox"/>
eric6-17.04.1.zip	<a href="#">2017-04-09</a>	20.8 MB	1,564 <input type="checkbox"/>
eric6-17.04.1.tar.gz	<a href="#">2017-04-09</a>	18.7 MB	456 <input type="checkbox"/>
eric6-nolang-17.04.1.zip	<a href="#">2017-04-09</a>	14.5 MB	3 <input type="checkbox"/>
eric6-nolang-17.04.1.tar.gz	<a href="#">2017-04-09</a>	12.3 MB	0 <input type="checkbox"/>
eric6-i18n-zh_CN-17.04.1.tar.gz	<a href="#">2017-04-09</a>	586.8 kB	8 <input type="checkbox"/>
eric6-i18n-zh_CN-17.04.1.zip	<a href="#">2017-04-09</a>	578.7 kB	52 <input type="checkbox"/>
eric6-i18n-tr-17.04.1.tar.gz	<a href="#">2017-04-09</a>	534.0 kB	0 <input type="checkbox"/>
eric6-i18n-tr-17.04.1.zip	<a href="#">2017-04-09</a>	521.9 kB	0 <input type="checkbox"/>
eric6-i18n-ru-17.04.1.tar.gz	<a href="#">2017-04-09</a>	912.2 kB	0 <input type="checkbox"/>

1-44

## 2. Eric 6

eric6-17.04.1 eric6 eric  
eric6 1-45

这台电脑 > 新加卷 (E:) > installed_software > eric6				
名称	修改日期	类型	大小	
eric	2017/3/22 18:17	文件夹		
changelog	2017/3/11 19:59	文件	13 KB	
install.py	2017/3/11 19:59	Python File	54 KB	
install-debugclients.py	2017/3/11 19:59	Python File	10 KB	
install-i18n.py	2017/3/11 19:59	Python File	4 KB	
LICENSE.GPL3	2017/3/11 19:59	GPL3 文件	33 KB	
patch_modpython.py	2017/3/11 19:59	Python File	5 KB	
README.rst	2017/3/11 19:59	RST 文件	11 KB	
README-i18n.rst	2017/3/11 19:59	RST 文件	1 KB	
THANKS	2017/3/11 19:59	文件	2 KB	
uninstall.py	2017/3/11 19:59	Python File	12 KB	
uninstall-debugclients.py	2017/3/11 19:59	Python File	3 KB	

图1-45

安装Eric 6时，运行pip install Qsci，出现“Error: cannot import name 'Qsci'”错误，如图1-46所示。

```
E:\install_software2\eric6-17.04.1> python install.py
Checking dependencies
Python Version: 3.5.3
Found PyQt5
Found pyuic5
Sorry, please install QScintilla2 and
its PyQt5/PyQt4 wrapper.
Error: cannot import name 'Qsci'
Press enter to continue..._
```

图1-46

安装Qsci，如图1-47所示。

运行命令：pip install QScintilla-i <https://pypi.douban.com/simple>



```

C:\windows\py.exe
Checking dependencies
Python Version: 3.5.3
Found PyQt5
Found pyuic5
Found QScintilla2
Found QtGui
Found QtNetwork
Found QtPrintSupport
Found QtSql
Found QtSvg
Found QtWidgets
Qt Version: 5.6.0
sip Version: 4.18
PyQt Version: 5.6
QScintilla Version: 2.9.2
All dependencies ok.

Cleaning up old installation ...

Creating configuration file ...

Compiling user interface files ...

Compiling source files ...

Installing eric6 ...

Installation complete.

Press enter to continue...

```

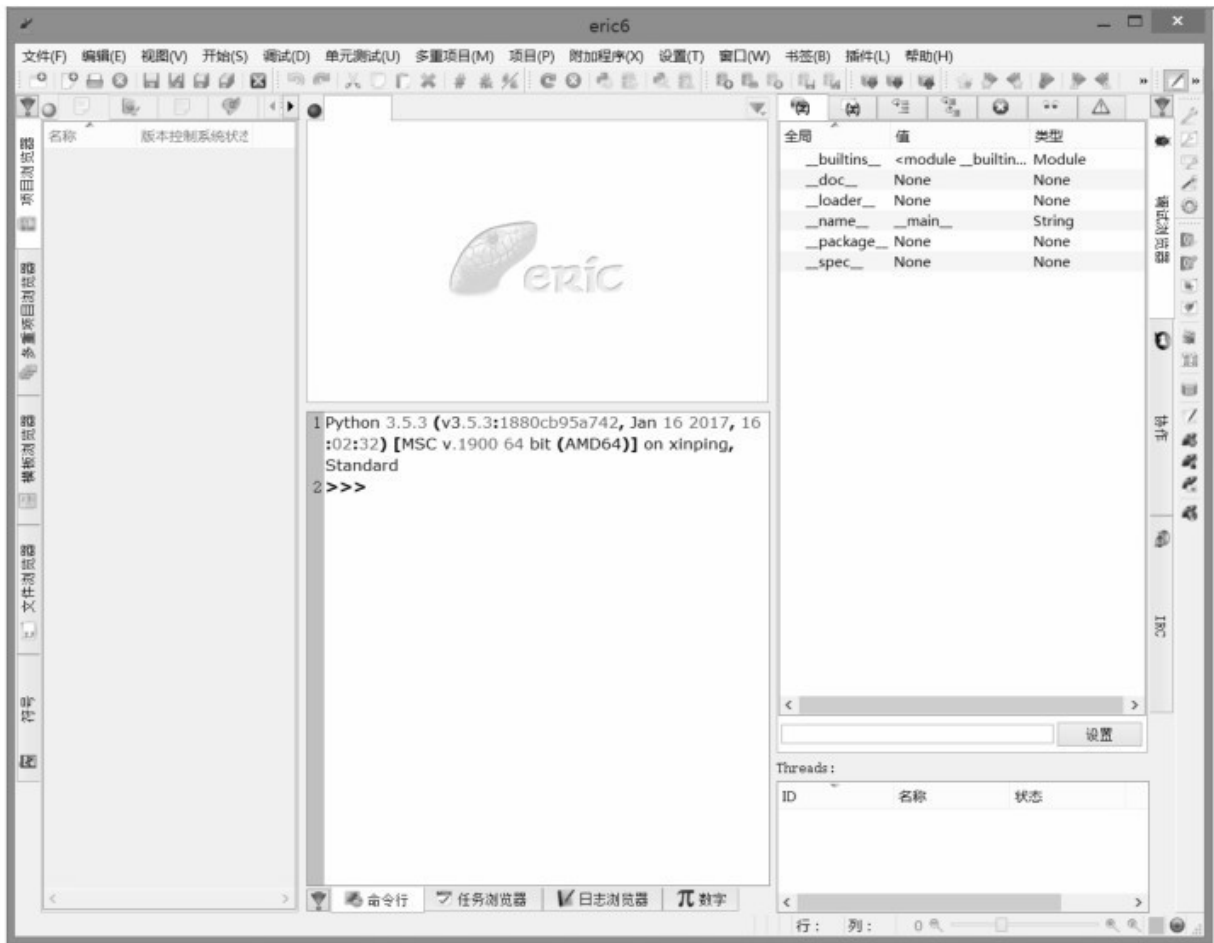
图1-48

zwPython > software > win-eric6 > eric6-17.01 > eric6-17.01 > eric

名称	修改日期	类型	大小
erico.appdata.xml	2014/1/1 12:23	XML 文档	3 KB
eric6.desktop	2015/1/18 11:56	DESKTOP 文件	1 KB
eric6.e4p	2016/12/31 13:42	E4P 文件	142 KB
eric6.py	2016/12/31 13:42	Python File	13 KB
eric6.pyw	2016/12/31 13:42	Python File (no c...	1 KB
eric6_api.py	2016/12/31 13:42	Python File	12 KB

图1-49

1-50



01-50

## 1.3.2 Eric 6

Eric 6 是一个基于 Qt 的 Python IDE，它集成了 Python 解释器、代码编辑器、调试器、任务浏览器、日志浏览器、数字计算器等工具。它支持 Python 2.7 和 Python 3.x 版本。

1. 安装 Qt 和 Qt Designer  
 2. 安装 pyqt5-tools  
 3. 安装 Qt 库  
 4. 安装 pip  
 5. 安装 Eric 6  
 6. 配置 Eric 6  
 7. 运行 Eric 6



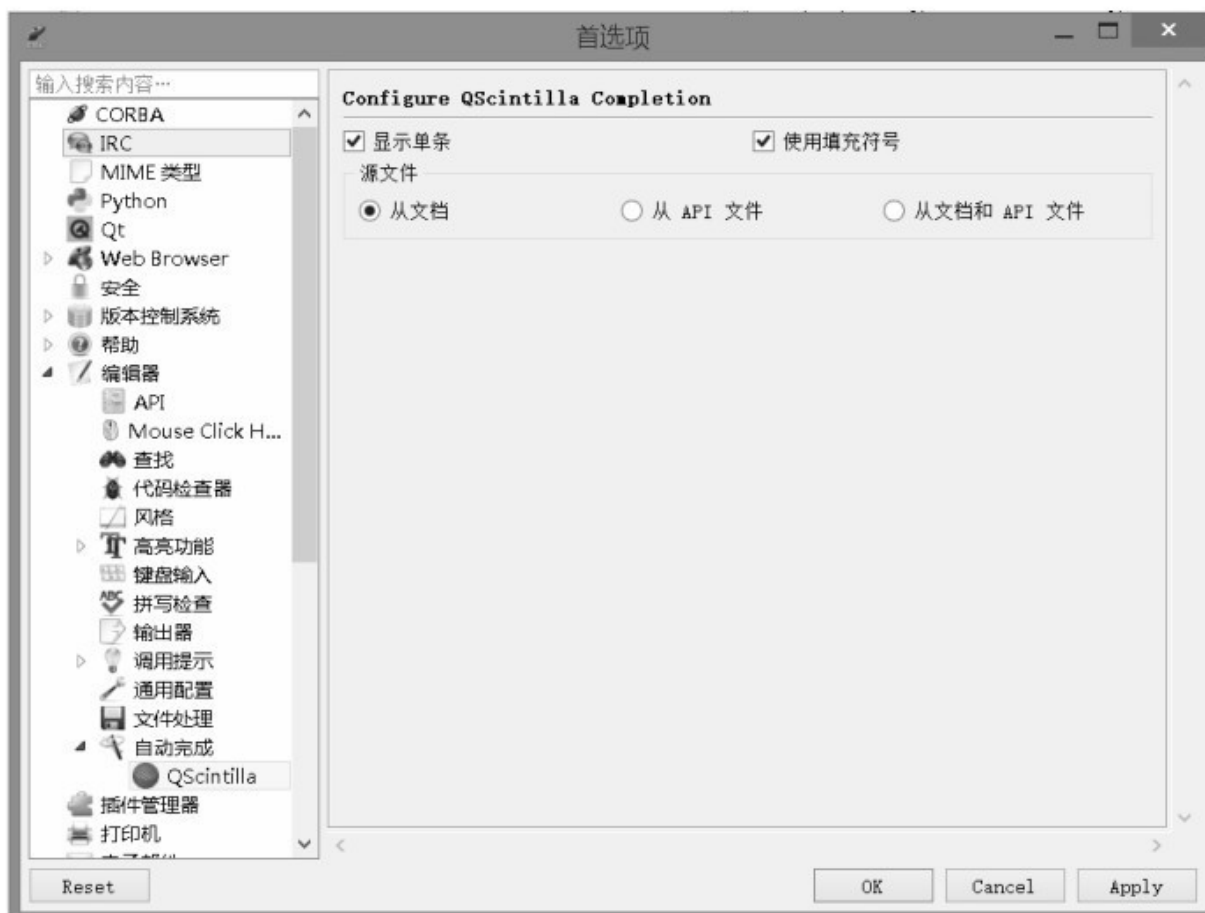


图 1-52

图 3 “”→“”“”1-53

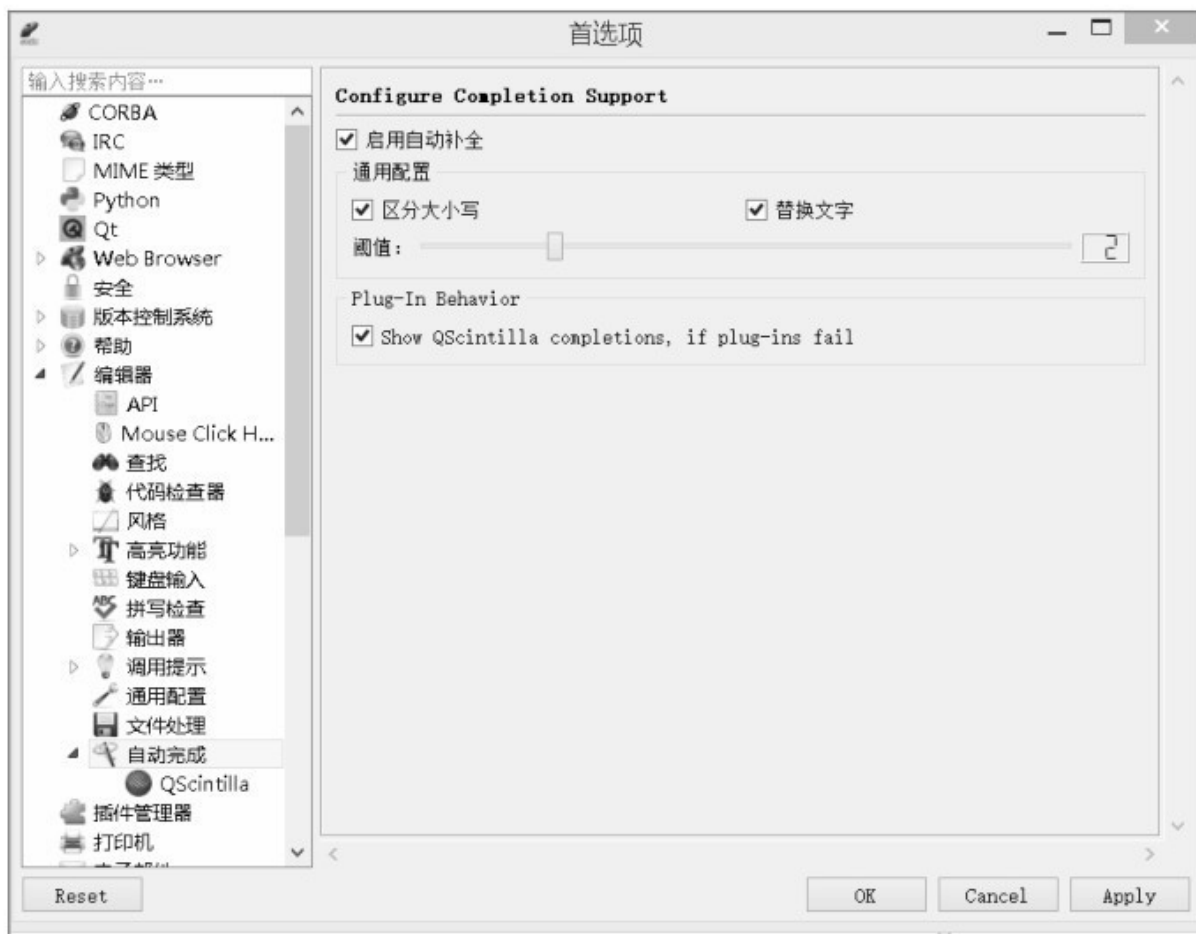


图1-53

在Eric 6中配置API

将“API”替换为“Python3”并设置API路径

在E:\installed\_software\python35\Lib\site-packages\PyQt5\qsci\api\python\eric6.api文件中，将“API”替换为“OK”



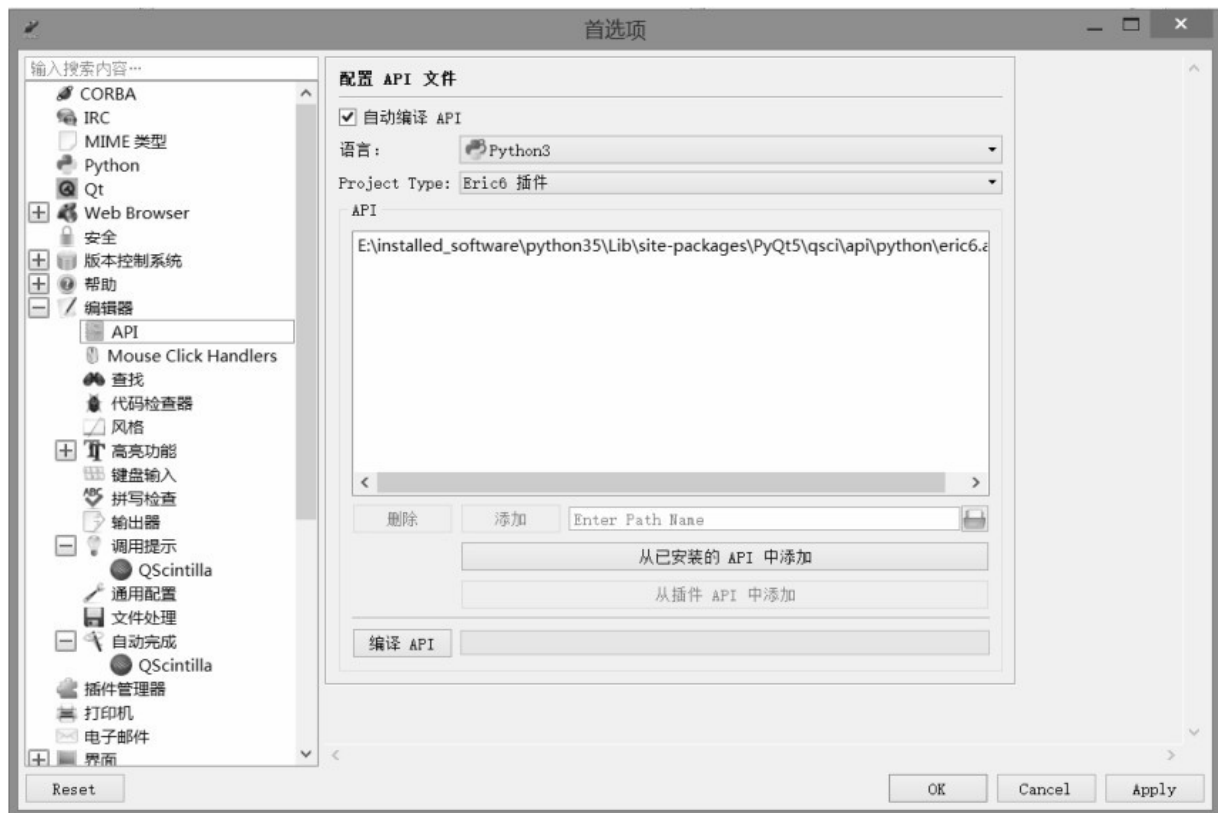


图 1-54

配置 PyQt 5 API 图 1-55



图1-55

Python 编码格式“utf-8” 1-56



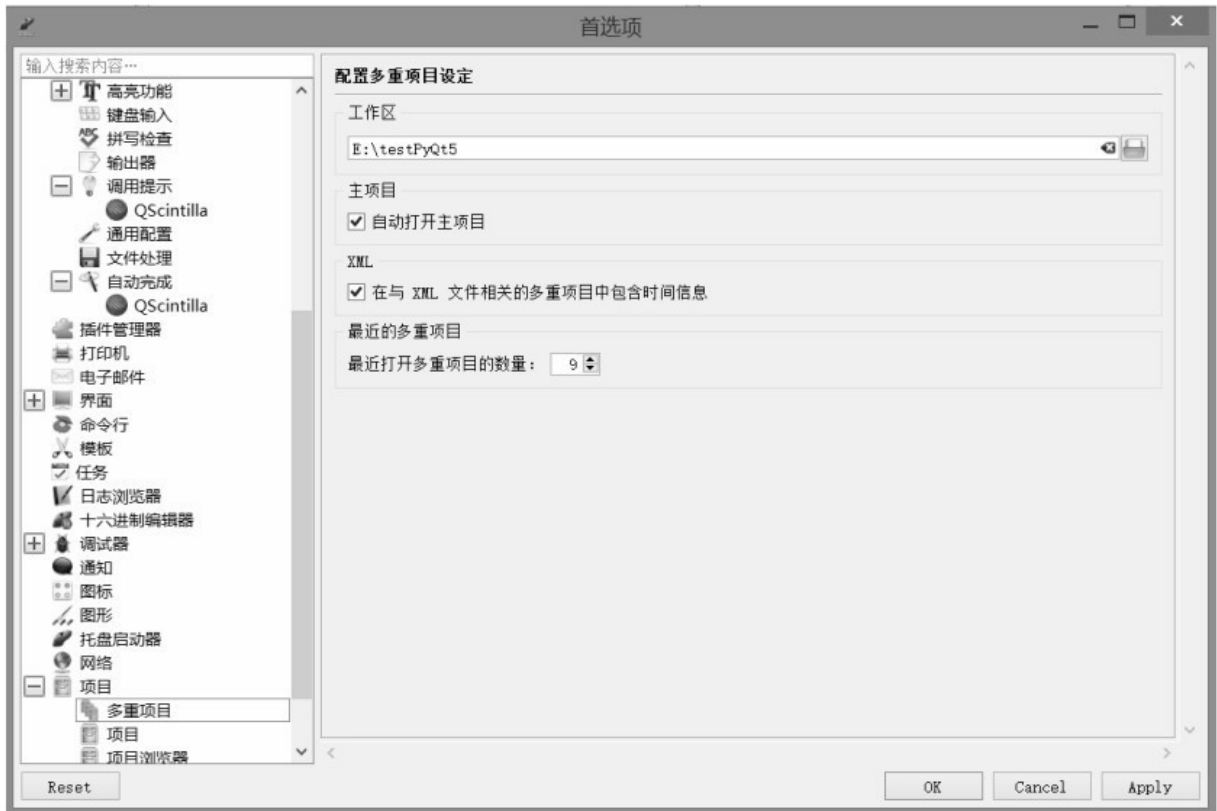


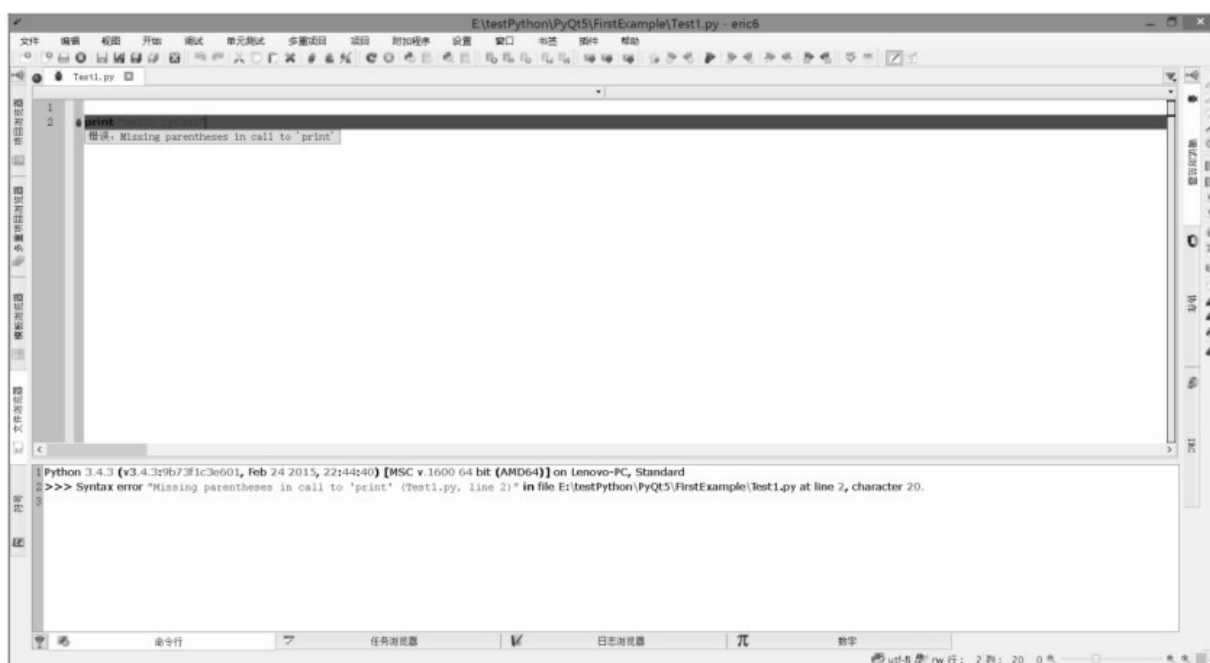
图1-57

在Eric 6的GUI中  
将“Initial zoom factor”设置为1.58



图1-58

图7 Eric 6 代码检查器 Eric 6 代码检查器 1-59



### 1.3.3 〇〇〇〇〇〇〇〇jedi

jedi Python IDE jedi  
Python jedi goto  
pydoc IDE

jedi□□□□□□□□□□□□□□□□Eric IDE□Vim□Emacs□Sublime  
Text □ TextMate □ Kate □ Atom □ SourceLair □ GNOME Builder □  
Visual Studio Code□Gedit□wdb□□

## 1. Jedi

□□□□□□jedi□

```
pip install jedi
```

□□□□□□□□□□1-60□□□□□□

```
D:\installed_software\Python34\Scripts>pip install jedi
Collecting jedi
  Downloading jedi-0.10.0-py2.py3-none-any.whl (186kB)
    100% |#####| 194k
Installing collected packages: jedi
Successfully installed jedi-0.10.0
```

## 2. Eric 6 jedi

Eric 6jediEric 6  
“”→“jedi1-611-64”

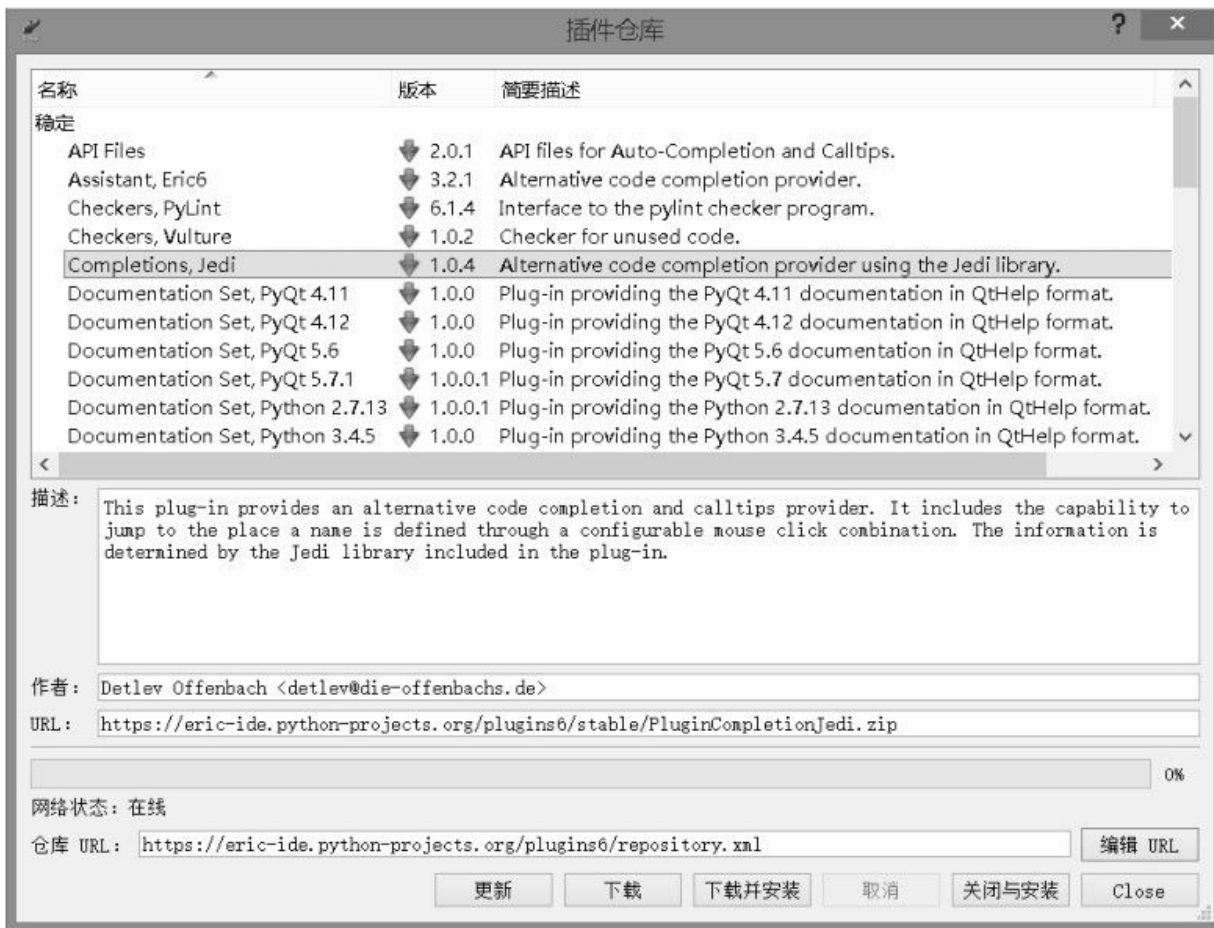


图1-61

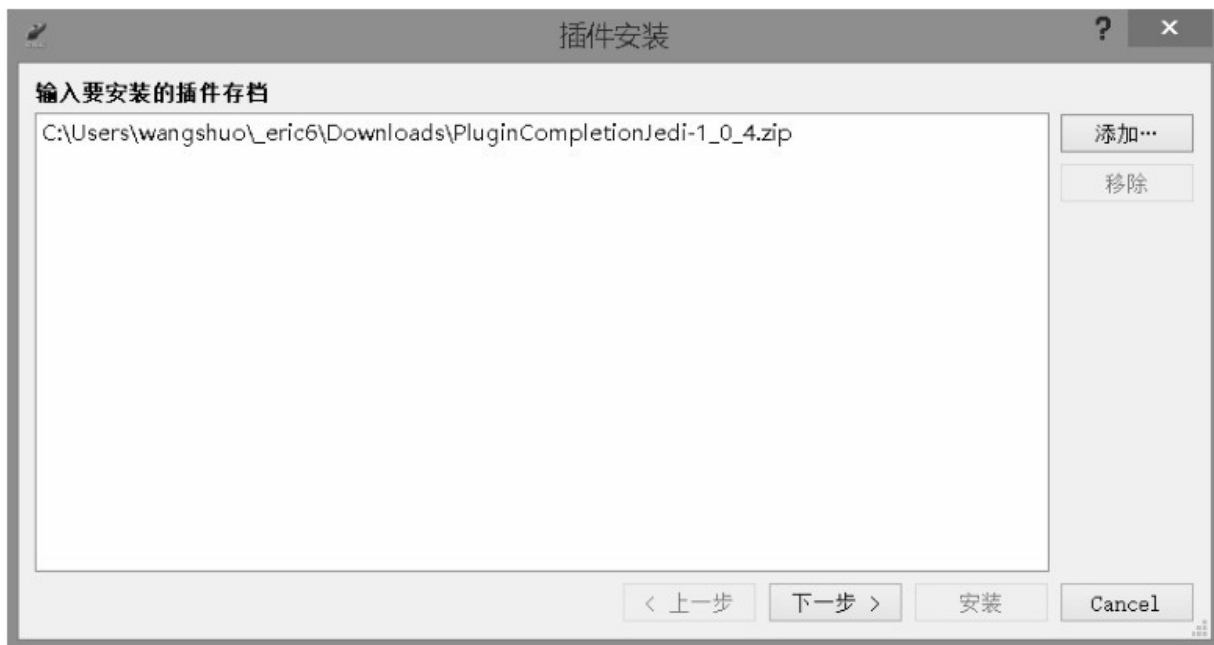


图1-62

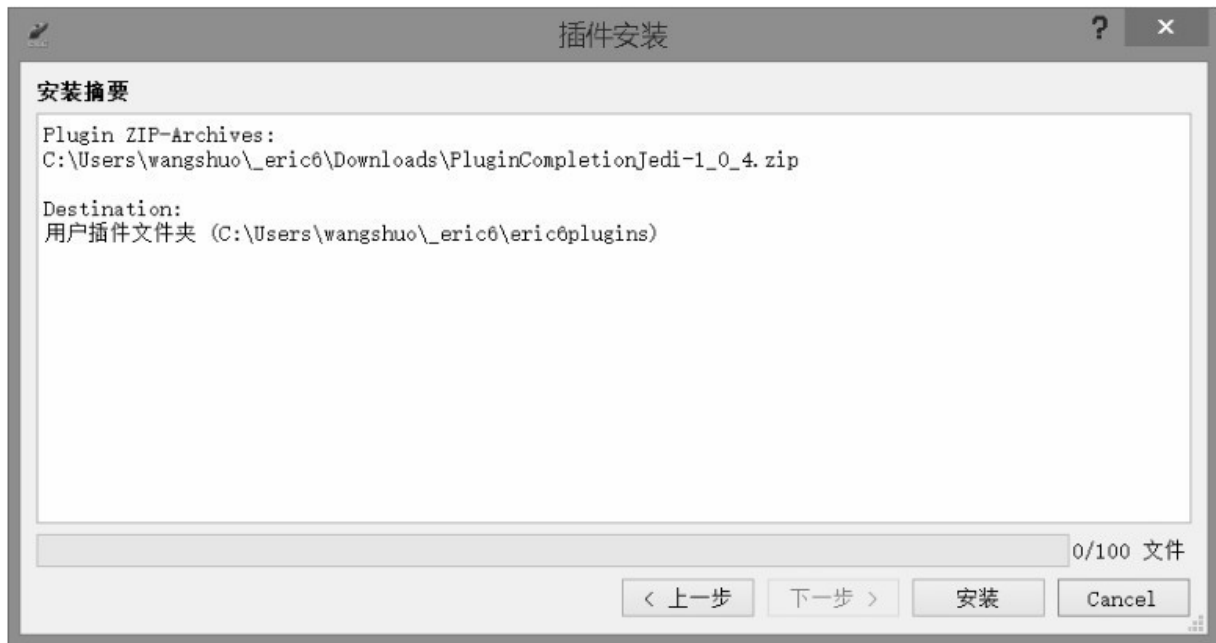


图1-63



图1-64

### 1.3.4 安装Eric 6



Eric 6 的 .py 文件 1-65 保存在 Eric 6 的  
目录

```
1 import sys
2 from PyQt5.QtWidgets import QWidget, QApplication
3
4 - if __name__ == '__main__':
5     app = QApplication(sys.argv)
6     q = QWidget()
7     q.show()
8     sys.exit(app.exec_()) |
```

图 1-65

### 1.3.5 Eric 6 的

Eric 6 的 PyQt 5 保存在  
PyQt5/Chapter01/EricPro01 目录  
图 1 的 PyQt 5 保存在 Eric 6 的 “” → “” 图 1-66

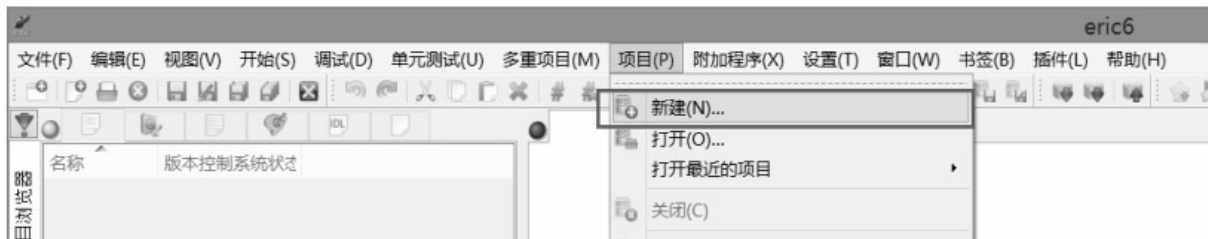


图 1-66

图 2 的 “OK” 图 1-67 保存在  
FirstPyQtPro  
PyQt5 GUI  
E:\quant\PyQt5\Chapter01\EricPro01 目录  
图 3

图 3 的 “No” 图 1-68

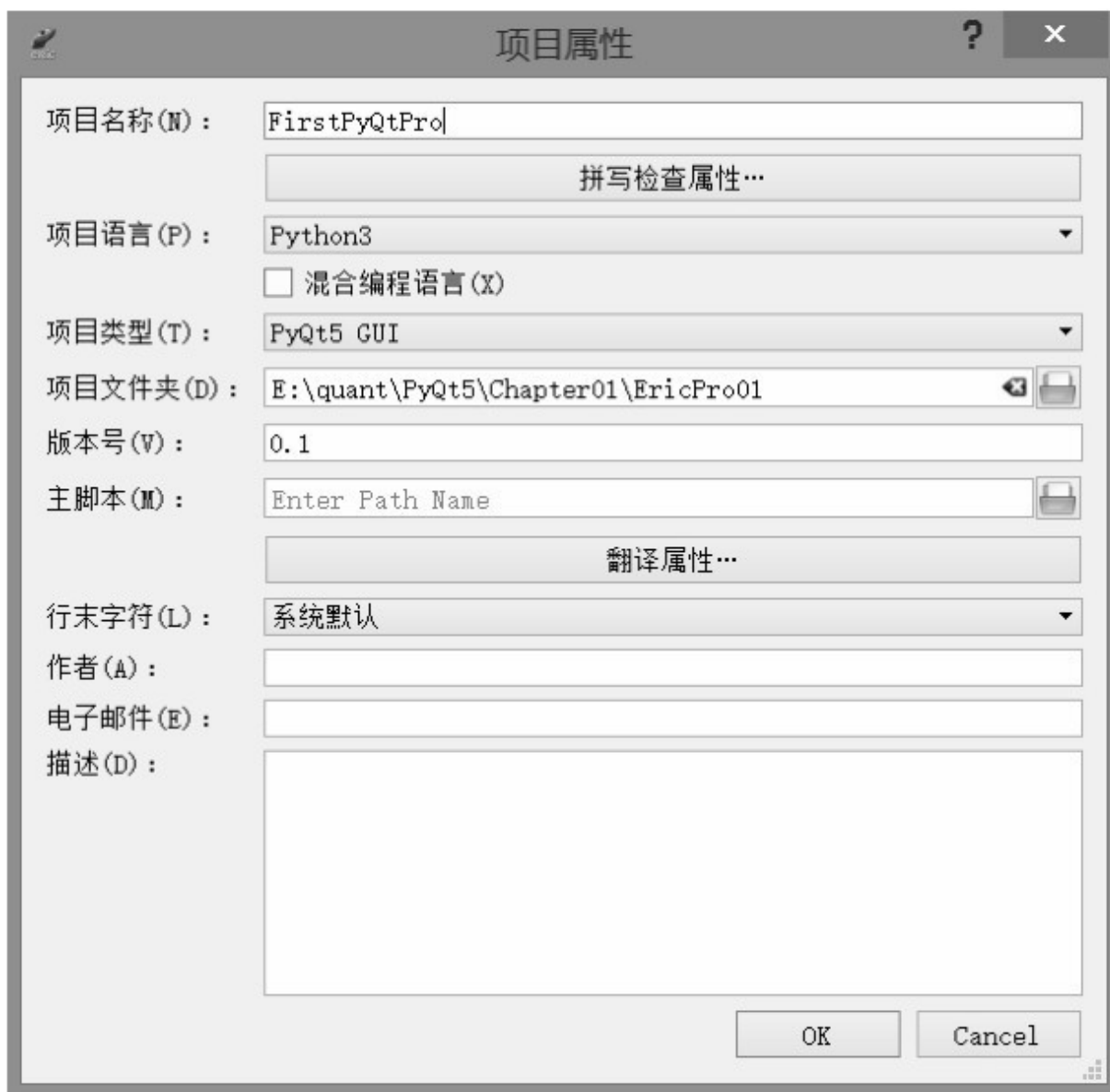


图1-67

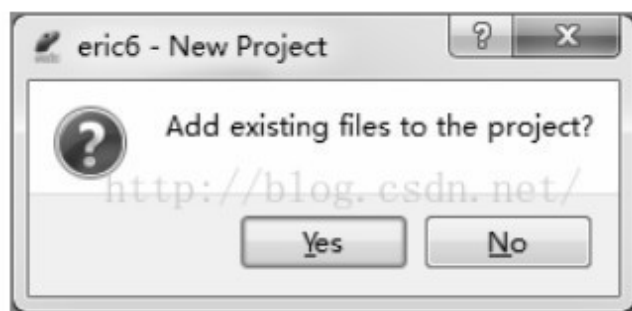


图1-68

第4步在 Python 解释器中将“”→“”Eric 解释器“”1”  
解释器解释器解释器“”1”1-69

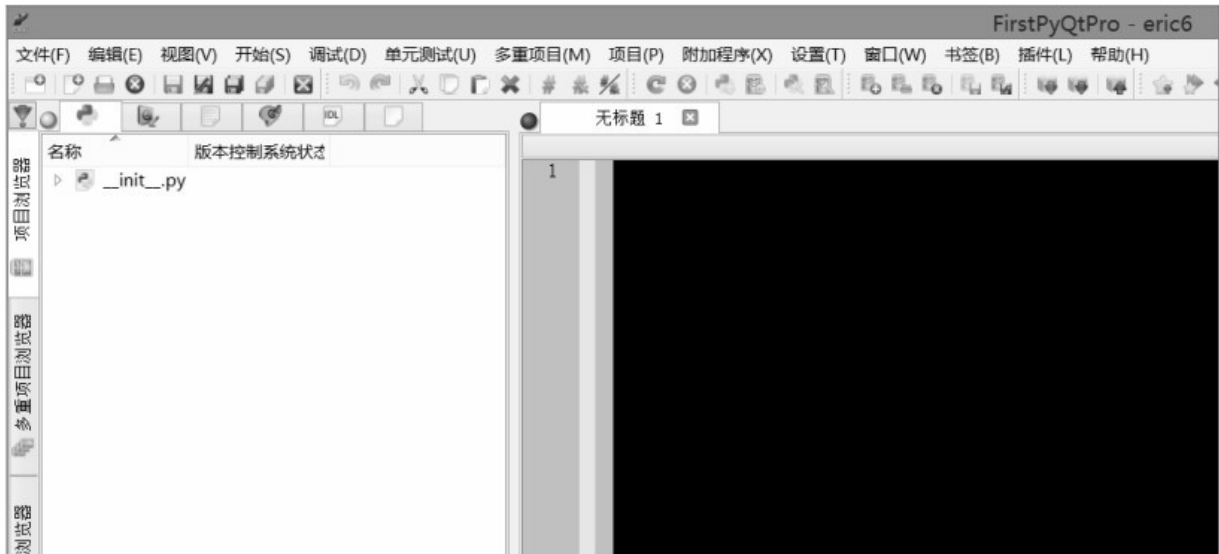


图1-69

第5步在“”解释器“”1”.py解释器1-70



图1-70

第6步在解释器解释器解释器 FirstWin.py解释器“Python3  
Files(\*.py)”解释器“”解释器1-71

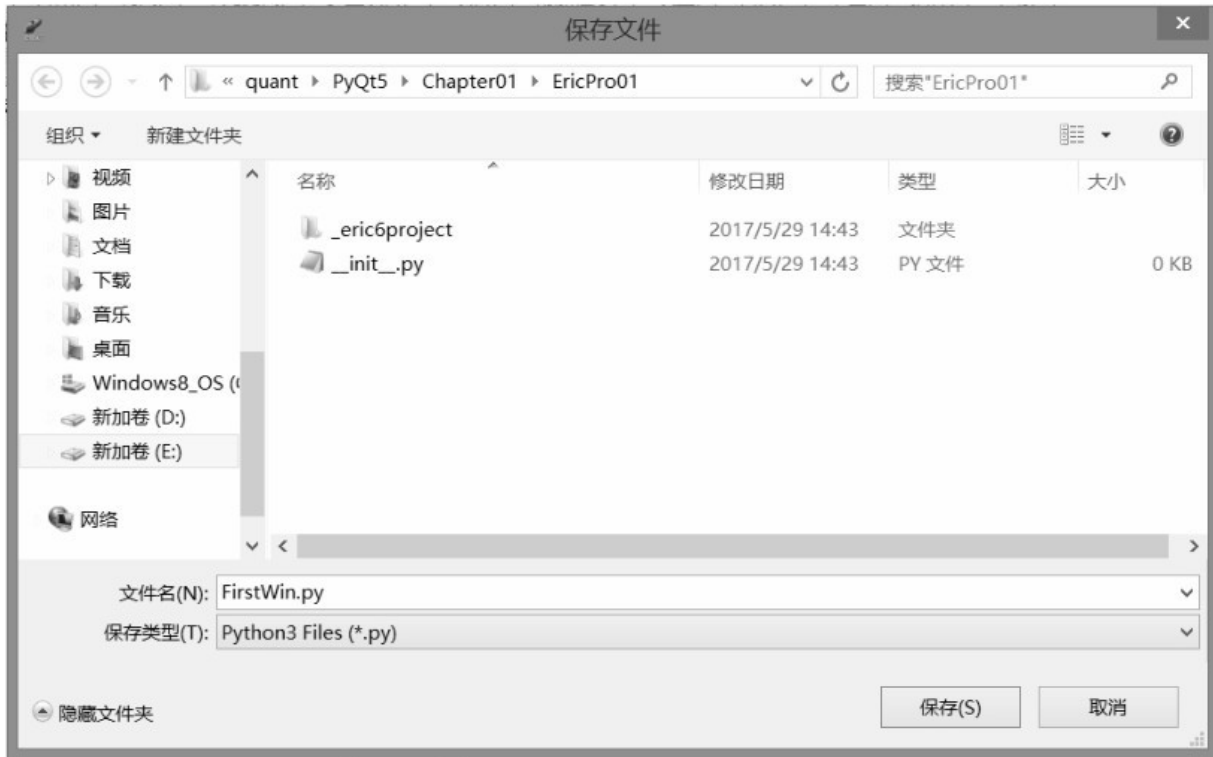


图1-71

“图1”所示的`FirstWin.py`文件如图1-72所示



图1-72

```

from PyQt5.QtWidgets import QApplication, QWidget, QPushButton
import sys

class WinForm( QWidget):
    def __init__(self,parent=None):

```

[illegible]

1-74



图1-74

## 1.4 安装Git

可以在GitHub上找到<https://github.com/cxinping/PyQt5>的源代码。

安装Git的步骤如下：

1. 下载Git。访问<https://git-scm.com/download/win>，下载Git-1.9.4-\*.exe。

2. 运行安装程序。将安装程序复制到E:/temp2，运行安装程序。在安装过程中，选择“Git Bash”选项。图1-75。



图1-75

在cmd窗口中执行

```
git clone https://github.com/cxinping/PyQt5.git
```

图1-76 克隆PyQt5仓库

```
E:\temp2> git clone https://github.com/templarXpWs/PyQt5.git
Cloning into 'PyQt5'...
remote: Counting objects: 13, done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 13 (delta 0), reused 13 (delta 0), pack-reused 0
Unpacking objects: 100% (13/13), done.
Checking connectivity... done.
```

图1-76

在本地计算机上创建名为PyQt5的文件夹，并设置其所有者为xpws2006@163.com。  
“PyQt5”文件夹  
在本地计算机上创建名为PyQt5的文件夹，并设置其所有者为xpws2006@163.com。

名称	修改日期	类型	大小
.git	2017/8/27 20:11	文件夹	
Chapter01	2017/8/7 10:26	文件夹	
Chapter02	2017/7/7 17:03	文件夹	
Chapter03	2017/8/7 10:26	文件夹	
Chapter04	2017/8/7 10:33	文件夹	
Chapter05	2017/8/24 20:50	文件夹	
Chapter06	2017/8/8 13:29	文件夹	
Chapter07	2017/8/7 10:37	文件夹	
Chapter08	2017/8/7 10:38	文件夹	
Chapter09	2017/8/27 19:54	文件夹	
Chapter10	2017/8/7 10:42	文件夹	
Chapter11	2017/8/27 14:21	文件夹	
tool	2017/8/27 19:46	文件夹	
README.md	2017/8/25 8:58	MD 文件	5 KB

图1-77

在本地计算机上创建名为PyQt5的文件夹，并设置其所有者为xpws2006@163.com。  
GitHub仓库  
在本地计算机上创建名为PyQt5的文件夹，并设置其所有者为xpws2006@163.com。  
git clone  
git pull  
PyQt5



- \PyQt5\Chapter\*\[empty]11[empty]PyQt 5 [empty]  
[empty]
- \PyQt5\tool\[empty] PyQt 5 [empty] Windows [empty] SQLite [empty]  
[empty]ClassGraphics.edx[empty]UML[empty]PyQt 5[empty]

## 2 Python

### 2.1 Python

Python 是一個簡單易學、強大且靈活的編程語言，在 IT 領域有著廣泛的應用。Python 是一個跨平台的編程語言，可以在 Windows、MacOS、Linux 等系統上運行。

1. Python 的安裝與配置：Python 的安裝過程非常簡單，只需下載並安裝 Python 解釋器即可。安裝後，需要配置環境變量，以便在命令行中直接運行 Python 程序。此外，還需要安裝一些常用的庫，如 pip、setuptools 等。

2. Python 的數據類型與變量：Python 支持多種數據類型，包括整數、浮點數、字符串、列表、字典等。變量的命名需要遵循一定的規則，如不能以數字開頭、不能包含特殊字符等。

- 安裝 pandas、NumPy 等數據科學庫。
- CUDA 加速 Python 程序，C/C++ 或 FORTRAN 與 NVIDIA 的集成。

3. Python 的應用場景：Python 可以用於多種場景，如數據分析、機器學習、Web 開發、遊戲開發等。

- 安裝 scikit-learn、Theano 等機器學習庫。
- 安裝 NLTK、spaCy 等自然語言處理庫。
- 安裝 Python、OpenCV 等圖像處理庫。

4. Python 的遊戲開發：

- 安裝 pygame 遊戲引擎，用於開發 2D 遊戲。
- 安裝 fontforge 字體編輯器，用於生成字體文件。

fonttools 是一個 Python 庫，用於處理字體文件。

- 安裝 Blender、GIMP、Inkscape、Maya、3D Max 等 3D 建模軟件。

Python 可以用於開發這些軟件的插件或腳本。

numpy pandas CUDA scikit-learn Theano pattern  
Python IT

Python Guido van Rossum ·  
Python

- 
- 
- 
- 

Python 8  
80 2011 1 TIOBE  
Python 2010

TIOBE  
Google MSN Yahoo!  
Wikipedia YouTube Baidu

Python  
List Dictionary Tuple 2017 6  
TOP20 Python Java C C++  
2-1 2017 6 “” Python



## 2.2 数据类型

Python 5 数据类型

- Number 数值
- String 字符串
- List 列表
- Tuple 元组
- Dictionary 字典

1

1 Python 数据类型 C 数据类型  $(-6+4j)$   $(5.3-7.6j)$

2 Python char 数据类型

数据类型

数据类型 Number 数据类型

$x=1$

$y=911$

Python 4 数据类型

- int 整数
- long 长整数
- float 浮点数
- complex 复数

Python 运算符

- + 加
- - 减
- \* 乘
- / 除

- // 整数除法
- % 取余数
- \*\* 幂运算

## 2-1 练习

在 2-1 文件夹下新建 py201math.py 文件，并打开  
 PyQt5/Chapter02/py201math.py 文件，用 Python 解释器运行该文件。

```
#1
print('\n#1')
x=10
y=22
z=35
print('x,y,z,',x,y,z)

#2
print('\n#2')
a=x+y;print('a=x+y,',a)
b=x-y;print('b=x-y,',b)
c=z-x*y;print('c=z-x*y,',c)

#3
print('\n#3')
a=z/x;print('a=z/x,',a)
b=z//x;print('b=z//x,',b)
c=z%x;print('c=z%x,',c)

#4
print('\n#4')
```







程序代码	对应的输出信息
<pre>#1 dss='  hello pyqt5,,' print("\n#1,去空格及特殊符号") s1=dss.strip().lstrip().rstrip(',') print('s1,',s1)</pre>	<pre>#1,去空格及特殊符号 s1, hello pyqt5</pre>
<pre>#2 print("\n#2,字符串连接") s2=dss.join(['a','.',',c']) print('s2,',s2) s3='s3' s3+='xx' print('s3,',s3)</pre>	<pre>#2,字符串连接 s2, a  hello pyqt5,..  hello pyqt5,,c s3, s3xx</pre>

程序代码	对应的输出信息
<pre>#3 print("\n#3,查找字符") css='abc1c2c3' pi=css.find('c') print(pi,pi)</pre>	<pre>#3,查找字符 pi, 2</pre>
<pre>#4,字符串比较 print("\n#4,字符串比较") print( s1 &gt; s2 ) print( s1 == s2 ) print( s1 &lt; s2 )</pre>	<pre>#4,字符串比较 True False False</pre>
<pre>#5 print("\n#5,字符串长度") s1,s2='abc','c123' print('len(s1),',len(s1)) print('len(s2),',len(s2))</pre>	<pre>#5,字符串长度 len(s1), 3 len(s2), 4</pre>
<pre>#6 print("\n#6,大小写转换") s1,s2='abc','ABC123efg' print('大写, s1.upper(),',s1.upper()) print('小写, s2.lower(),',s2.lower()) print('大小写互换 ,s2.swapcase(),',s2.swapcase()) print('首字母大写 ,s1.capitalize(),',s1.capitalize())</pre>	<pre>#6,大小写转换 大写, s1.upper(), ABC 小写, s2.lower(), abc123efg 大小写互换 ,s2.swapcase(), abc123EFG 首字母大写 ,s1.capitalize(), Abc</pre>
<pre>#7 print("\n#7,分割字符串") s2=' hello, ziwang.com,,' print('s2.split(),s2.split(',')')</pre>	<pre>#7,分割字符串 s2.split, [' hello', ' ziwang', 'com', ', ', '']</pre>

## 2.4 List□□□□



```
zlst,['hello','PyQt5','.', 'com']
```

```
vlst,['Top','Quant','.', 'vip']
```

```
#2
```

```
s2,['PyQt5','.', 'com']
```

```
s3,['PyQt5','.']
```

```
s4,['Top','Quant','.']
```

```
#3
```

```
s2+s3,['PyQt5','.', 'com','PyQt5','.']
```

```
s3*2,['PyQt5','.', 'PyQt5','.']
```

```
□□□+□□□□□□□□□□□□*□□□□□□□□
```

```
□□□□□□□□□□□□
```

```
□1□□□□□□□□□□□□
```

- `cmp(list1,list2)`□□□□□□□□□□

- `len(list)`□□□□□□□□

- `max(list)`□□□□□□□□□□□□

- `min(list)`□□□□□□□□□□□□

- `list(seq)`□□□□□□□□□□

```
□2□□□□□□□□□□□□
```

- `list.append(obj)`□□□□□□□□□□□□

- `list.count(obj)`□□□□□□□□□□□□□□

- `list.extend(seq)`□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
□□□
```

- `list.index(obj)`□□□□□□□□□□□□□□□□□□□□

- `list.insert(index,obj)`□□□□□□□□□□

- `list.pop(obj=list[-1])`□□□□□□□□□□□□□□□□□□□□□□□□□□

```
□□□□
```

- `list.remove(obj)`□□□□□□□□□□□□□□□□

- list.reverse() 反轉list
- list.sort([func]) 對list排序

## 2.5 Tuple

元組“( )” 由多個字元或字串所組成，且元組中的字元或字串不可修改。  
 元組

### 範例2-5

範例 2-5 元組的應用 py205tuple.py 檔案位於  
 PyQt5/Chapter02/py205tuple.py

```
#1
print('\n#1')
zlst=('hello','PyQt5','.','com')
vlst=('Top','Quant','.','vip')
print('zlst,',zlst)
print('vlst,',vlst)

#2
print('\n#2')
s2=zlst[1:];print('s2,',s2)
s3=zlst[1:3];print('s3,',s3)
s4=vlst[:3];print('s4,',s4)

#3
print('\n#3')
print('s2+s3,',s2+s3)
```

```

print('s3*2,',s3*2)
#####
#1
zlst,('hello','PyQt5','.', 'com')
vlst,('Top','Quant','.', 'vip')
#2
s2,('PyQt5','.', 'com')
s3,('PyQt5','.')
s4,('Top','Quant','.')
#3
s2+s3,('PyQt5','.', 'com','PyQt5','.')
s3*2,('PyQt5','.', 'PyQt5','.')

```

## 2.6 Dictionary

字典“{ }”由键key和值value组成。Python  
字典由键值对 k-v 组成。字典是可变容器。  
字典

Python 字典由键值对组成。字典是可变容器。  
字典

字典由键值对组成。字典是可变容器。key  
字典

### 2-6

□ □ 2-6 □ □ □ □ py206dict.py □ □ □ □ □ □  
PyQt5/Chapter02/py206dict.py□□□□□□□□□□□□□□□□

#1

print('\n#1')

zdict={}

zdict['w1']='hello'

zdict['w2']='ziwang.com'

print('zdict,',zdict)

#2

print('\n#2')

vdic={ 'url1':'TopQuant.vip'

      ,'url2':'www.TopQuant.vip'

      ,'url3':'ziwang.com'}

print('vdic,',vdic)

#3

print('\n#3')

s2=zdict['w1'];print('s2,',s2)

s3=vdic['url2'];print('s3,',s3)

□□□□□□□□□□

#1

zdict,{'w2': 'ziwang.com','w1': 'hello'}

#2

vdic,{'url3':                              'ziwang.com','url2':

'www.TopQuant.vip','url1':'TopQuant.vip'}

#3

s2,hello

s3,www.TopQuant.vip





- `long(x [,base])`將x轉換為long型
- `float(x)`將x轉換為float型
- `complex(real)`將real轉換為complex型
- `str(x)`將x轉換為string型
- `repr(x)`將x轉換為repr型
- `eval(str)`將str轉換為Python可執行的程式碼
- `tuple(s)`將s轉換為tuple型
- `list(s)`將s轉換為list型
- `chr(x)`將x轉換為character型
- `unichr(x)`將x轉換為Unicode型
- `ord(x)`將x轉換為integer型
- `hex(x)`將x轉換為hexadecimal型
- `oct(x)`將x轉換為octal型

## 2.8 控制結構

Python提供了多種控制結構，包括 `if...else`、`while`、`for` 等。

### 2-7 控制結構

在 2-7 節中，我們將使用 `py207ctrl.py` 文件來演示 Python 的控制結構。該文件位於 `PyQt5/Chapter02/py207ctrl.py`。Python 的控制結構包括：

```
#1 使用 if-else 結構
print('\n1,if')
```

```

x,y,z=10,20,5
if x<y:
    print('x<y')
else:
    print('x>y')
#####
1,if
x<y
2#####elif#####
#2
print('\n#2,elif')
x,y,z=10,20,5
if x<y:
    print('x<y')
elif x<z:
    print('x<z')
#####
#2,elif
x<z
3#####while#####
#3
print('\n#3,while')
x=3
while x>0:
    print(x)
    x-=1
#####

```

### #3,while

3

2

1

4 for

## #4

```
print('\n#4,for')
```

```
xlst=['1','b','xxx']
```

```
for x in xlst:
```

```
print(x)
```

[illegible]

## #4,for

1

**b**

5 for

## #5

```
print('\n#5,for')
```

```
for x in range(3):
```

```
print(x)
```

□ □ □ □ □ □ □ □ □

## #5,for

0

1

2

Python for

[illegible]

## 2.9 関数

Python 関数とは、特定の処理を実行するためのコードの塊を指す。関数は、特定のタスクを自動化し、コードの再利用を促進する。関数は、引数を受け取り、結果を返すことができる。関数の定義は、`def` キーワードで始まる。

### 例2-8 関数

例 2-8 は、`py208fun.py` というファイルに記述されている。このファイルは `PyQt5/Chapter02/py208fun.py` に保存されている。

```
def f01(a,b,c):  
    print('a,b,c,',a,b,c)  
    a2,b2,c2,=a+c,b*2,c*2  
    return a2,b2,c2
```

```
#1  
print('\n#1')  
x,y,z=f01(1,2,3)  
print('x,y,z,',x,y,z)  
#2  
print('\n#2')  
x,y,z=f01(x,y,z)  
print('x,y,z,',x,y,z)
```

実行結果

```
#1  
a,b,c,1 2 3  
x,y,z,4 4 6  
#2
```



```
plus5=functools.partial(add,5)
```

```
#2
```

```
print('\n#2')
```

```
rst2=plus3(4)
```

```
print('plus3(4)=' ,rst2)
```

```
rst3=plus3(7)
```

```
print('plus3(7)=' ,rst3)
```

```
rst4=plus5(10)
```

```
print('plus5(10)=' ,rst4)
```

```
#####
```

```
#1
```

```
add(4,2)=6
```

```
#2
```

```
plus3(4)=7
```

```
plus3(7)=10
```

```
plus5(10)=15
```

```
#####
```

Python#####

```
plus3=functools.partial(add,3)
```

```
####'3'#####add()#####plus3()
```

```
rst3=plus3(4)
```

```
#####'7'##### plus3()#####'3'#####
```

3+4=7

```
plus3(7)=10
```

```
#####'7'#####plus3()#####3+7=10
```

```
#####
```

● partial#####

- partial

## 2.11 lambda

lambda는 lambda를 사용하여 Python에서 def를 사용하여 정의된 함수와 유사하게 함수를 정의할 수 있습니다. lambda는 Python에서 def를 사용하여 정의된 함수와 유사하게 함수를 정의할 수 있습니다.

### 예제 2-10 lambda

예제 2-10은 py210fun.py 파일에 PyQt5/Chapter02/py210fun.py에 lambda를 사용하여 함수를 정의하는 예제입니다.

```
fun1=lambda x,y : x + y
print('fun1(2,3)=',fun1(2,3))
fun2=lambda x: x*2
print('fun2(4)=',fun2(4) )
```

실행 결과

```
fun1(2,3)=5
fun2(4)=8
```

lambda는 Python에서 def를 사용하여 정의된 함수와 유사하게 함수를 정의할 수 있습니다. lambda는 Python에서 def를 사용하여 정의된 함수와 유사하게 함수를 정의할 수 있습니다.

## 2.12

```

class
    1
    2
    MyClass.count
    self
    self.name

```

## 2-11

2-11 py211class.py PyQt5/Chapter02/py211class.py

```

class MyClass:
    count=0
    name='DefaultName'
    def __init__(self,name):
        self.name=name
        print(' %s\n %s' % (
MyClass.name,self.name) )
    def setCount(self,count ):
        self.count=count
    def getCount(self):
        return self.count
if __name__=="__main__":

```



```

cls=MyClass('lisi')
cls.setCount(10)
print('count=%d' % cls.getCount())
"""
"""
DefaultName
lisi
count=10
__init__ Python  __init__
"""
"""
setCount()getCount()

```

## 2.13

```

def
self
__private_method
self.__private_methods
__private_attrs
self.__private_attrs

```

## 2-12

2-12 py212privateProperty.py  
PyQt5/Chapter02/py212privateProperty.py

□□□□□□□□

```
class MyCounter:
    __secretCount=0 # □□□□
    publicCount=0 # □□□□
    def __privateCountFun(self):
        print('□□□□□□□')
        self.__secretCount +=1
        self.publicCount +=1
        #print (self.__secretCount)
    def publicCountFun(self):
        print('□□□□□□□')
        self.__privateCountFun()
if __name__=="__main__":
    counter=MyCounter()
    counter.publicCountFun()
    counter.publicCountFun()
    print          ('instance          publicCount=%d'          %
counter.publicCount)
    print          ('Class          publicCount=%d'          %
MyCounter.publicCount)
□□□□□□□□
    □□□□□□□
    □□□□□□□
    □□□□□□□
    □□□□□□□
instance publicCount=2
Class publicCount=0
```



□□□□□□□□□□

□ □ 2-13 □ □ □ □ py213property01.py □ □ □ □ □ □  
PyQt5/Chapter02/py 213property01.py□□□□□□□□□□□□□□□□  
□□□□□□□□□□

```
class MyClass(object):
    def __init__(self):
        self._param=None
    def getParam(self):
        print( "get param: %s" % self._param)
        return self._param
    def setParam(self,value):
        print( "set param: %s" % self._param )
        self._param=value
    def delParam(self):
        print( "del param: %s" % self._param)
        del self._param
    param=property(getParam,setParam,delParam)
if __name__=="__main__":
    cls=MyClass()
    cls.param=10
    print("current param : %s " % cls.param )
    del cls.param
```

□□□□□□□□

```
set param: None
get param: 10
current param : 10
del param: 10
```

property(getx,setx,deltx) 实现  
getter/setter

**@property** 实现

MyClass 继承 object 实现  
\_param 使用 @property 实现 Python 的“属性”访问  
setter/getter 实现

2-13 py213property02.py  
PyQt5/Chapter02/py213 property02.py

```
class MyClass(object):
    def __init__(self):
        self._param=None
    @property
    def param(self):
        print( "get param: %s" % self._param)
        return self._param
    @param.setter
    def param(self,value):
        print( "set param: %s" % self._param )
        self._param=value
    @param.deleter
    def param(self):
        print( "del param: %s" % self._param)
        del self._param
if __name__=="__main__":
    cls=MyClass()
    cls.param=10
    print("current param : %s " % cls.param )
```

```
del cls.param
```

□□□□□□

```
set param: None
```

```
get param: 10
```

```
current param : 10
```

del param: 10

□ □

## 第3章 Qt Designer

本章介绍UI设计工具Qt Designer，它是PyQt 5的重要组成部分，用于设计Qt GUI。Qt Designer可以生成UI文件，这些文件可以与Python代码一起使用，以创建GUI应用程序。

### 3.1 Qt Designer

Qt Designer是Qt框架的一部分，用于设计GUI。它允许用户通过拖放控件来设计界面，并生成UI文件。PyQt框架包含Qt Designer的Python绑定，使得在Python中使用Qt Designer成为可能。UI文件通常以.ui格式保存，并可以与Python代码（.py文件）一起使用。Python代码通常使用Eric 6等工具来编写和运行。图3-1展示了Qt Designer的界面。

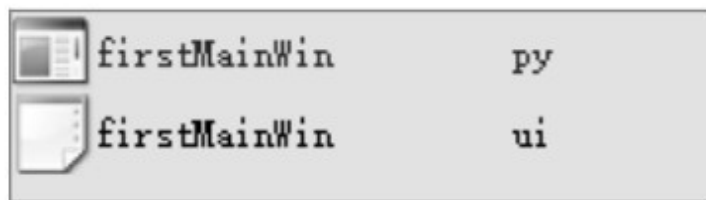


图3-1

Qt Designer采用MVC模式——模型——视图——控制器模式来设计GUI。

Qt Designer的界面如下：

- 设计器界面由以下部分组成：





3-4 创建“Main Window”工程，工程名为firstMainWin.ui

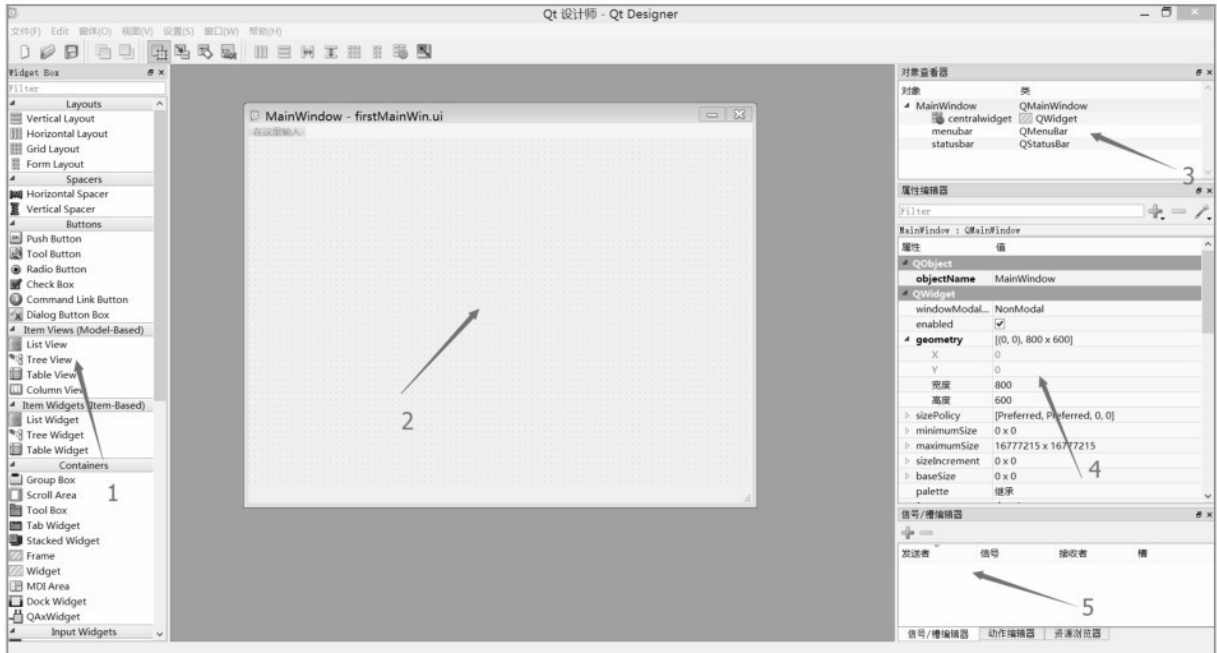


图3-4

### 3.1.2 创建主窗口

图3-4所示的Qt Designer界面中，1处Widget Box中的List View widget，2处是主窗口的画布，3处是对象查看器，4处是属性编辑器，5处是信号/槽编辑器。在Qt Designer中，可以通过“文件”菜单中的“新建”选项来创建一个新的Qt Designer项目。在弹出的“新建”对话框中，选择“Qt Designer”项目类型，然后点击“确定”按钮。此时，Qt Designer会创建一个新的主窗口，并显示图3-4所示的界面。

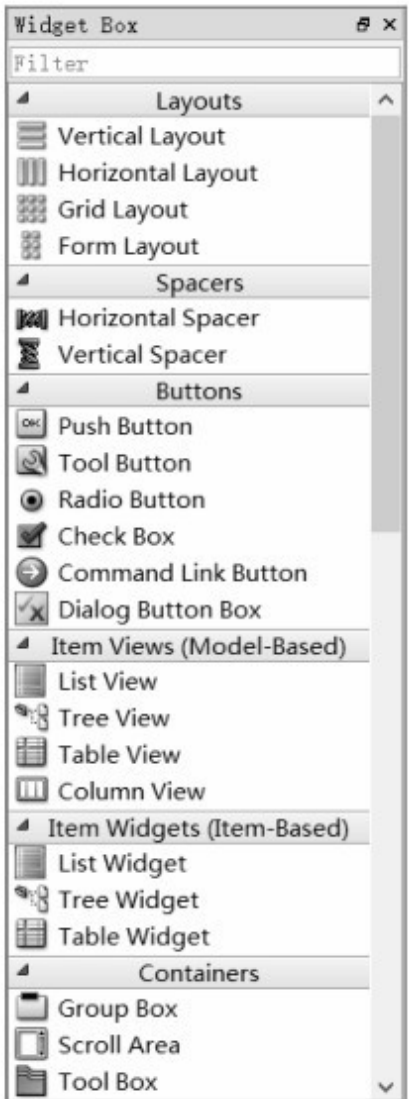


图3-5

Buttons 2 3-6

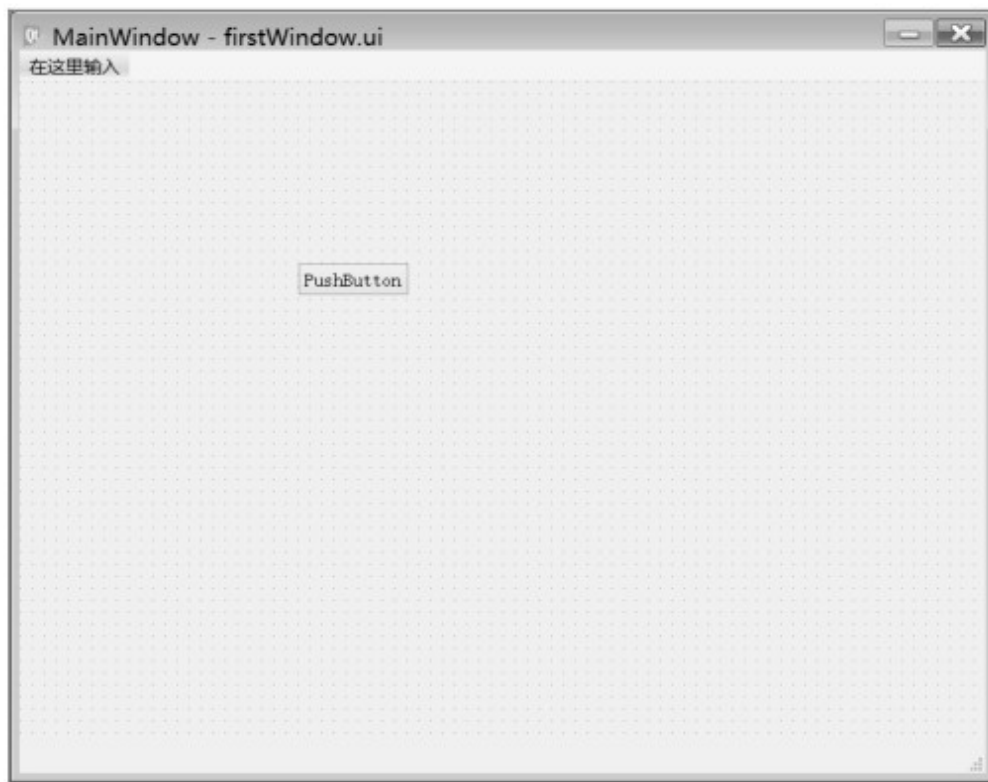


图 3-6

图 3-7 显示了 Qt Designer 中的对象查看器 (Object Inspector)。

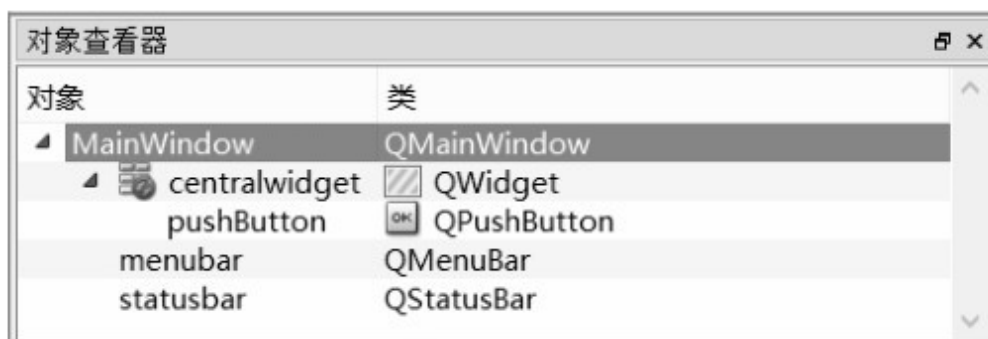


图 3-7

图 3-8 显示了 Qt Designer 中的信号槽 (Signals and Slots) 配置界面。

图 3-8

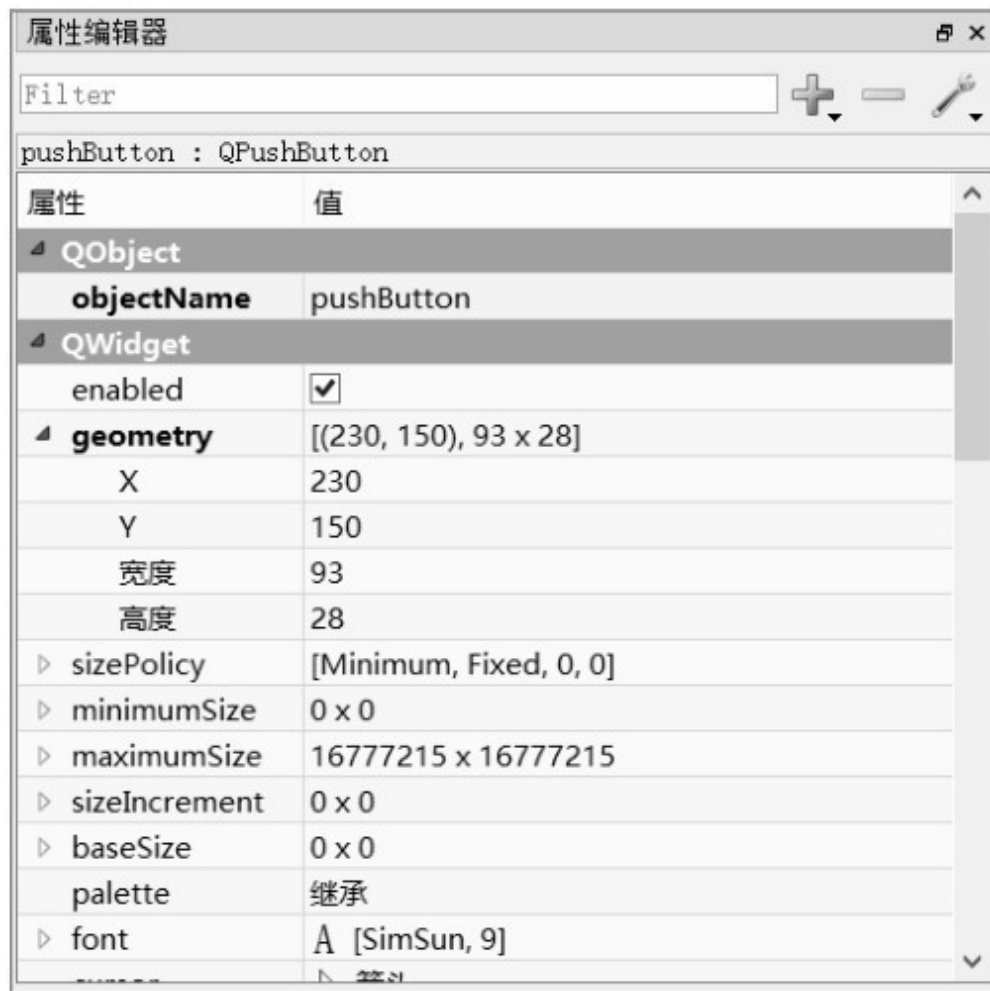
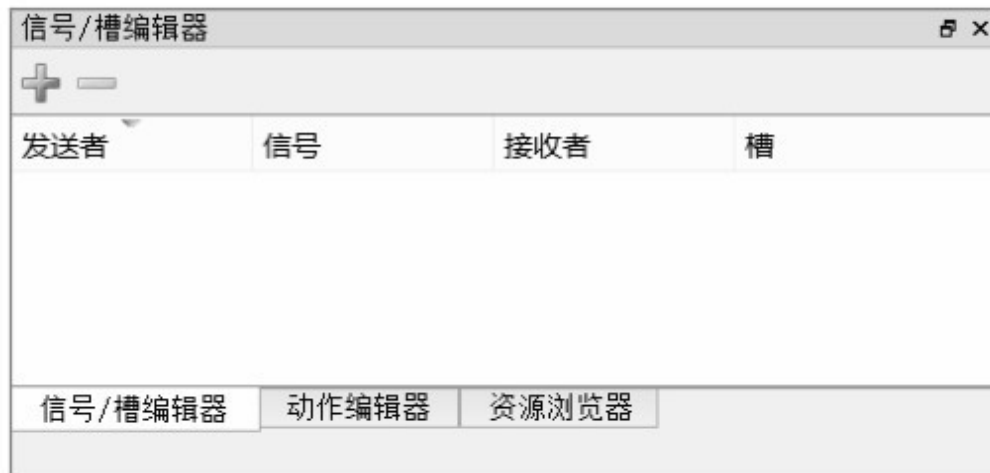


图3-8

- objectName 对象名称
  - geometry 几何形状
  - sizePolicy 大小策略
  - minimumSize 最小尺寸
  - maximumSize 最大尺寸
- minimumSize 和 maximumSize 用于指定控件的最小和最大尺寸。
- font 字体
  - cursor 光标
  - windowTitle 窗口标题

- windowsIcon/iconPath/Path
- iconSize/Size
- tooltip/Tooltip
- statusTip/statusTip
- text/text
- shortcut/Shortcut

5/ /  
3-9



□3-9

Label Button 3-10



图3-10

### 3.1.3 创建UI

在 Qt Designer 中新建一个名为 firstMainWin.ui 的 UI 文件，该文件将使用 XML 格式存储 UI 设计。

在 Qt Designer 中打开 PyQt5/Chapter03/firstMainWin.ui 文件，在画布上添加一个 QPushButton 对象，其 objectName 属性设置为 "pushbutton"，位置为 (490, 110)，宽度为 93px，高度为 28px，如图 3-11 所示。

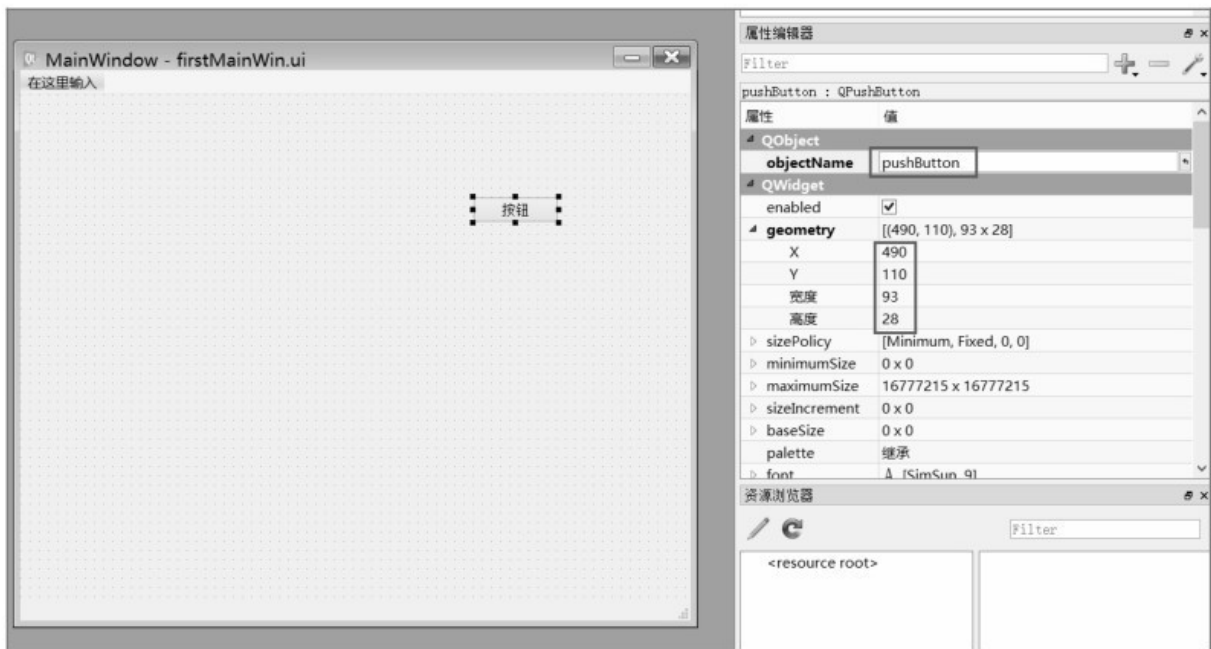


图3-11

保存文件 firstMainWin.ui，如图 3-12 所示。

```
firstMainWin.ui
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ui version="4.0">
3   <class>MainWindow</class>
4   <widget class="QMainWindow" name="MainWindow">
5     <property name="geometry">
6       <rect>
7         <x>0</x>
8         <y>0</y>
9         <width>726</width>
10        <height>592</height>
11      </rect>
12    </property>
13    <property name="windowTitle">
14      <string>MainWindow</string>
15    </property>
16    <widget class="QWidget" name="centralwidget">
17      <widget class="QPushButton" name="pushButton">
18        <property name="geometry">
19          <rect>
20            <x>490</x>
21            <y>110</y>
22            <width>93</width>
23            <height>28</height>
24          </rect>
25        </property>
26        <property name="text">
27          <string>按钮</string>
28        </property>
29      </widget>
30    </widget>
31    <widget class="QMenuBar" name="menubar">
32      <property name="geometry">
33        <rect>
34          <x>0</x>
35          <y>0</y>
36          <width>726</width>
37          <height>26</height>
38        </rect>
39      </property>
40    </widget>
41    <widget class="QStatusBar" name="statusbar"/>
42  </widget>
43  <resources/>
44  <connections/>
45 </ui>
```



圖 3-12 在 Qt Designer 中建立 .ui 檔案  
Qt Designer 可以將 UI 設計轉換為 XML 檔案，並生成對應的 Python 檔案。  
圖 3-13 顯示了 Eric 6 中 .ui 檔案的生成過程。

### 3.1.4 將 .ui 檔案轉換為 .py 檔案

Qt Designer 生成的 .ui 檔案是 XML 格式，需要通過 Python 腳本轉換為 Python 檔案。  
圖 3-13 顯示了 Eric 6 中 .ui 檔案的生成過程。

#### 1. 在 Eric 6 中生成 .ui 檔案

在 Eric 6 中，通過菜單「File」->「New」->「Form」可以生成新的 .ui 檔案。  
圖 3-13 顯示了 Eric 6 中 .ui 檔案的生成過程。

圖 3-13

在 Eric 6 中，通過菜單「File」->「New」->「Form」可以生成新的 .ui 檔案。  
圖 3-13 顯示了 Eric 6 中 .ui 檔案的生成過程。

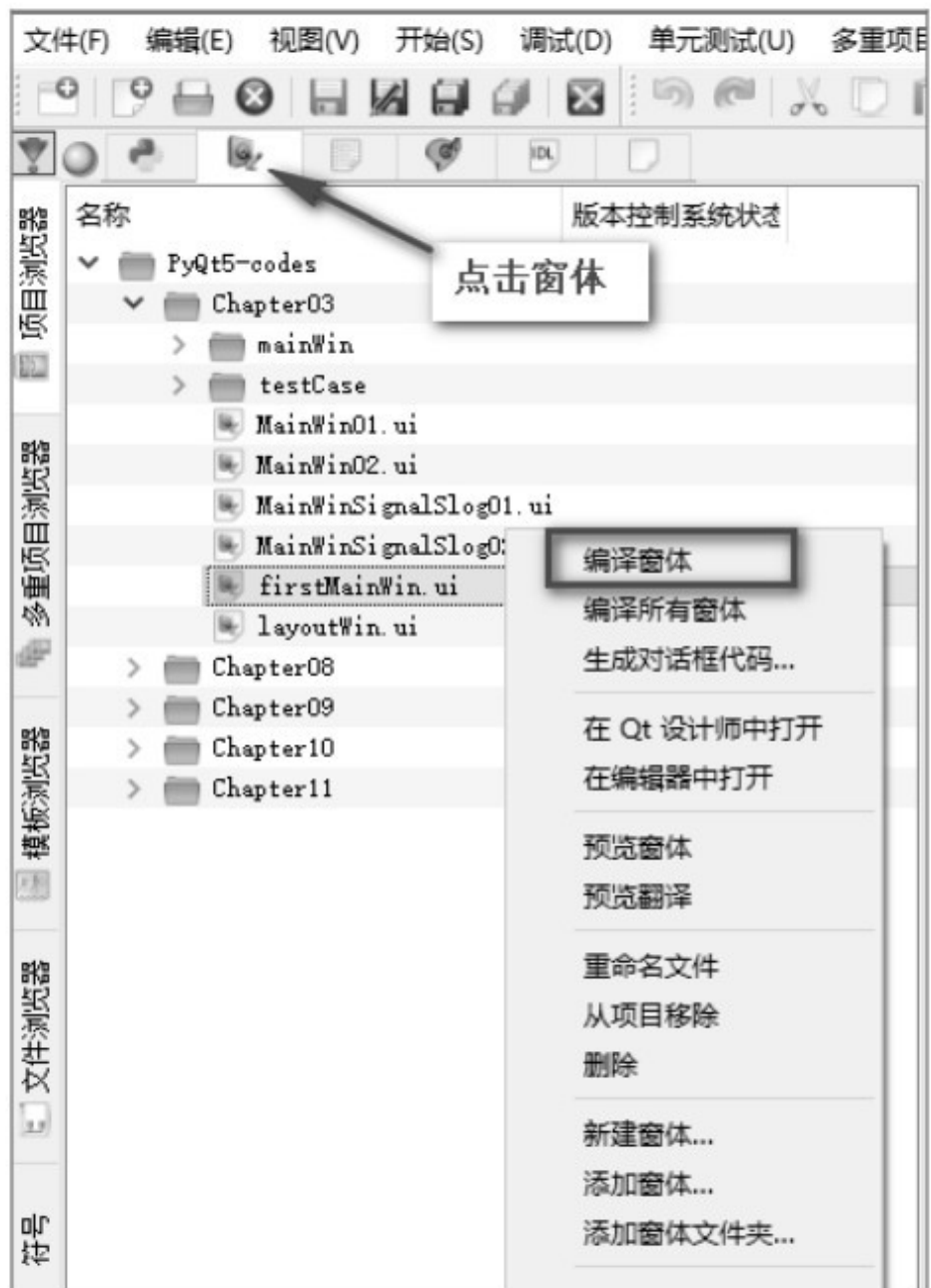


图3-13

在图3-14中，我们看到了在Qt Designer中创建的主窗口Ui\_firstMainWin.py文件。



图3-14

在 Ui\_firstMainWin.py 文件中将“”→“”并添加F2  
 图3-15 firstMainWin.ui 文件中的.ui文件  
 .py文件

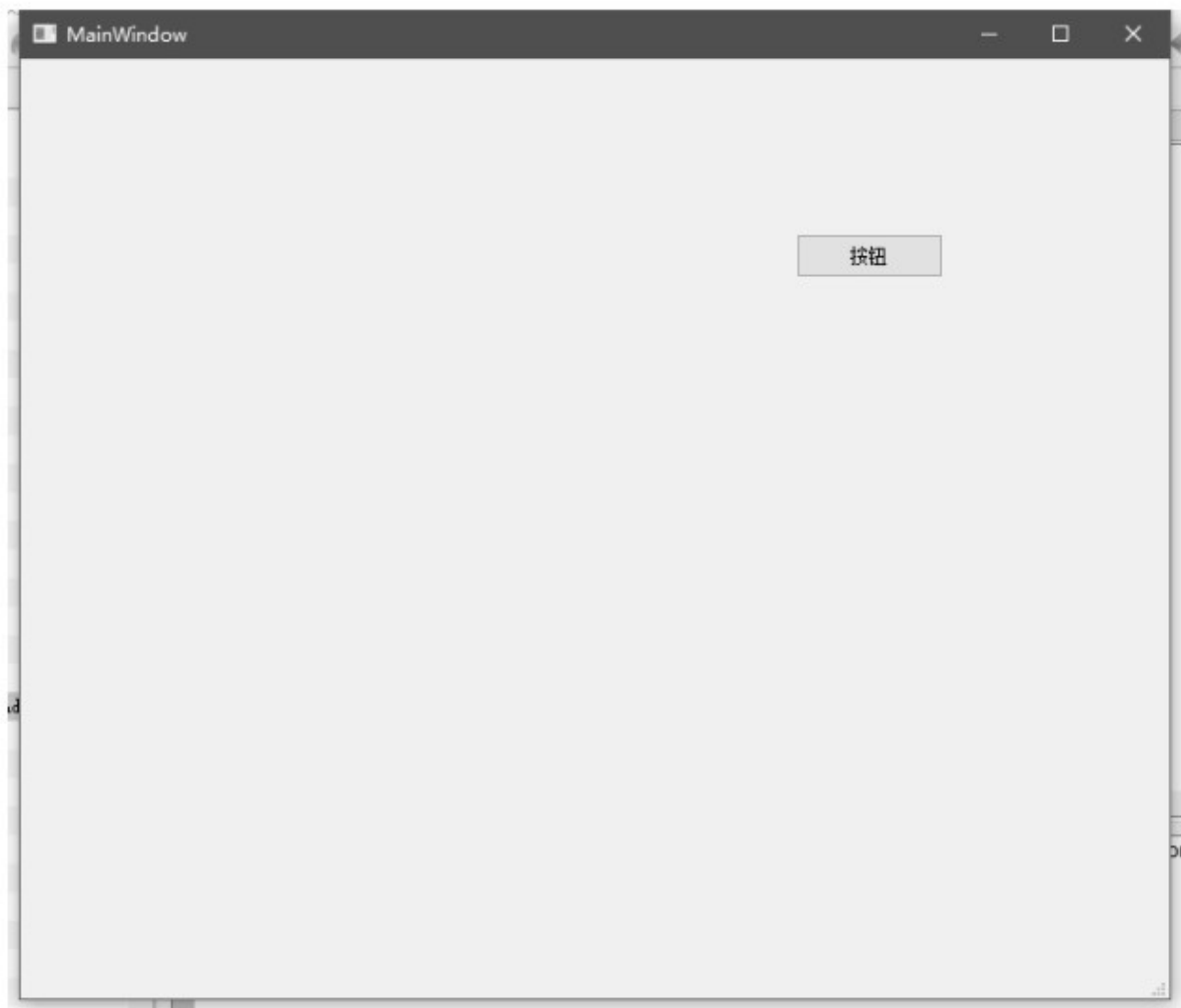


图3-15

## 2. 创建firstMainWin.ui和firstMainWin.py

PyQt 5使用pyuic5将.ui文件转换为.py文件。在Windows系统中，pyuic5通常位于Python安装目录的Scripts子目录下。例如，在E:\installed\_software\python35\Scripts\pyuic5.exe。

在命令行中，可以使用以下命令将firstMainWin.ui文件转换为firstMainWin.py文件：

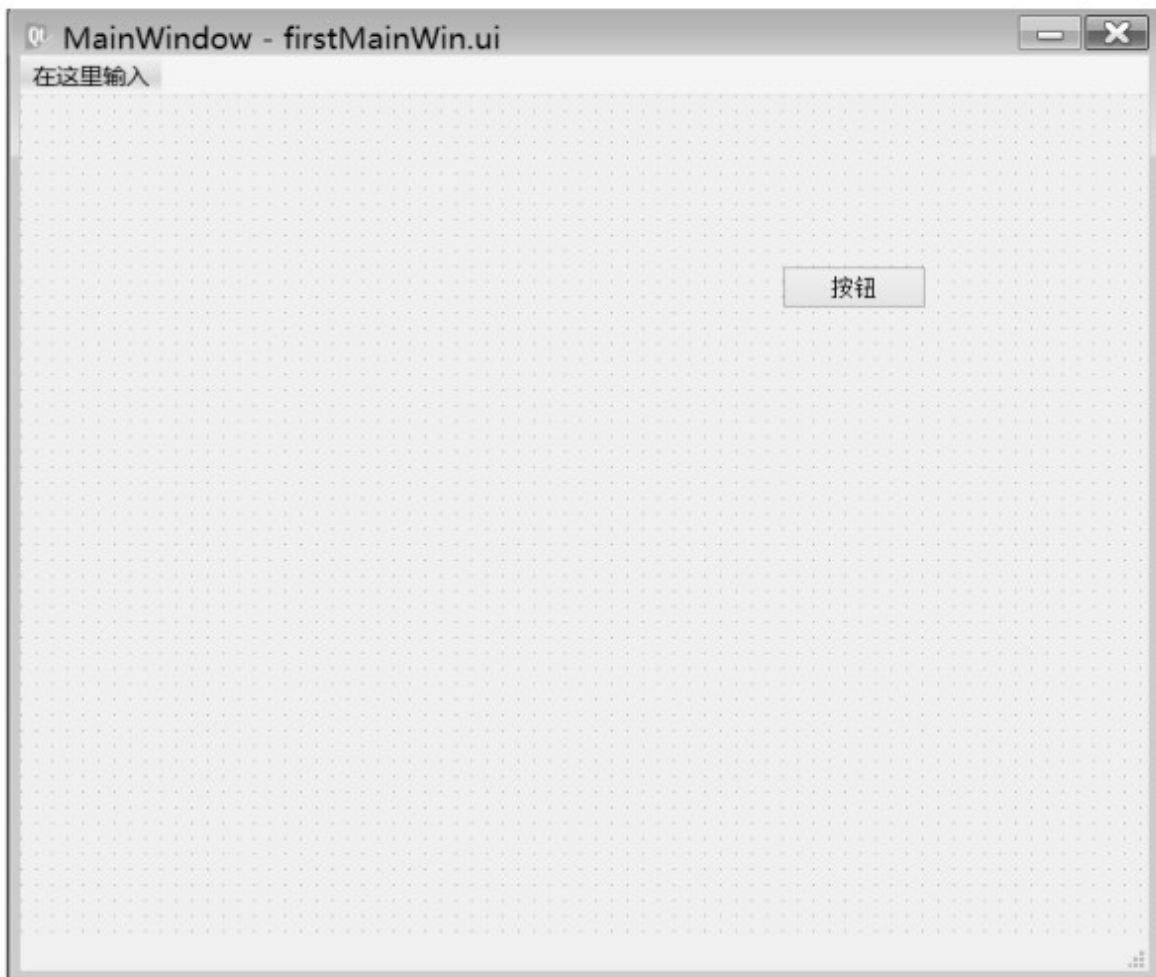
```
pyuic5 firstMainWin.ui -o firstMainWin.py
```

生成的firstMainWin.py文件是一个Python脚本，它使用PyQt 5的UI模块来加载firstMainWin.ui文件，并创建了一个名为firstMainWin的窗口对象。这个窗口对象是UI设计器中定义的窗口的Python表示。





Qt Designer 3-18 firstMainWin.ui



3-18

在  
firstMainWin.py Ui\_firstMainWin.py  
firstMainWin.py

### 3.1.5

.ui .ui .py  
.py .ui .py .py

mainwindow.ui를 mainwindow.py로 저장하고, mainwindow.py를 firstMainWin.py로 저장하고, firstMainWin.py를 CallFirstMainWin.py로 저장하고, CallFirstMainWin.py를 실행하면 됩니다.

```
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow
from firstMainWin import *
class MyMainWindow(QMainWindow, Ui_MainWindow):
    def __init__(self, parent=None):
        super(MyMainWindow, self).__init__(parent)
        self.setupUi(self)
if __name__ == "__main__":
    app = QApplication(sys.argv)
    myWin = MyMainWindow()
    myWin.show()
    sys.exit(app.exec_())
```

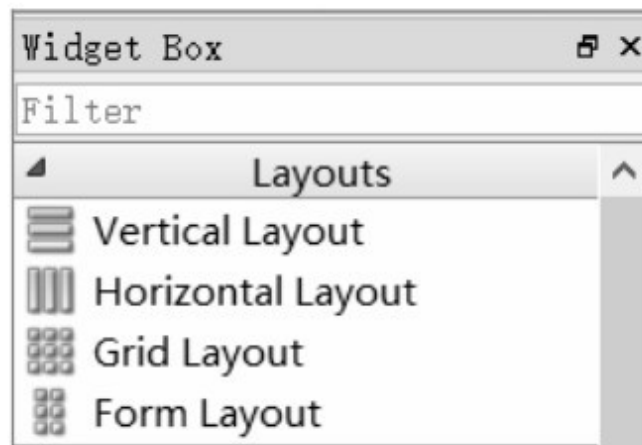
mainwindow.ui 파일을 mainwindow.py로 저장하고, firstMainWin.py 파일을 firstMainWin.py로 저장하고, firstMainWin.py 파일을 CallFirstMainWin.py로 저장하고, CallFirstMainWin.py 파일을 실행하면 됩니다.

## 3.2 PyQt5



3.1 Qt Designer의 Widget Box  
Qt Designer의 Widget Box는 Qt Designer의 상단에 위치하며, 다양한 Qt Widget과 Layout을 제공하는 도구 상자입니다.

Qt Designer의 Widget Box에는 4개의 주요 Layout이 포함되어 있습니다: Vertical Layout, Horizontal Layout, Grid Layout, Form Layout. 이 Layout들은 Qt Designer의 Widget Box에서 쉽게 접근할 수 있으며, 각각의 Layout은 3-19에 설명되어 있습니다.



3-19

- Qt Designer의 Widget Box에서 Layout을 선택하는 방법
  - Qt Designer의 Widget Box에서 Layout을 사용하는 방법
  - Qt Designer의 Widget Box에서 Layout을 사용하는 방법 (row, column, cell을 사용하여)
- Qt Designer의 Widget Box에서 Layout을 사용하는 방법은 다음과 같습니다:
- Qt Designer의 Widget Box에서 Layout을 선택하는 방법

### 3.2.1 Qt Designer의 Widget Box

Qt Designer의 Widget Box에서 QLineEdit과 QPushButton을 사용하는 방법은 다음과 같습니다:

Form - MainWin01.ui\*

确定

- 改变文本...
- 改变对象名称...
- 变型为
- 改变工具提示...
- 改变“这是什么”...
- 改变样式表...
- 大小限定
- 提升的窗口部件...
- 放到后面(B) Ctrl+K
- 放到前面(F) Ctrl+L
- 剪切(T) Ctrl+X
- 复制(C) Ctrl+C
- 粘贴(P) Ctrl+V
- 选择全部(A) Ctrl+A
- 删除(D)
- 布局
  - 调整大小(S) Ctrl+J
  - 水平布局(H) Ctrl+1
  - 垂直布局(V) Ctrl+2
  - 使用分裂器水平布局(P) Ctrl+3
  - 使用分裂器垂直布局(L) Ctrl+4
  - 栅格布局(G) Ctrl+5
  - 在窗体布局中布局(F) Ctrl+6
  - 打破布局(B)
  - 简易网格布局(M)

```

.ui .py
PyQt5/Chapter03/MainWin01.py
from PyQt5 import QtCore,QtGui,QtWidgets

```

```

class Ui_Form(object):
    def setupUi(self,Form):
        Form.setObjectName("Form")
        Form.resize(511,443)
        self.widget=QtWidgets.QWidget(Form)
        self.widget.setGeometry(QtCore.QRect(50,40,273,300))
        self.widget.setObjectName("widget")
        self.horizontalLayout=QtWidgets.QHBoxLayout(self.widget)
        self.horizontalLayout.setContentsMargins(0,0,0,0)
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.lineEdit_2=QtWidgets.QLineEdit(self.widget)
        self.lineEdit_2.setObjectName("lineEdit_2")
        self.horizontalLayout.addWidget(self.lineEdit_2)
        self.pushButton_2=QtWidgets.QPushButton(self.widget)
        self.pushButton_2.setObjectName("pushButton_2")
        self.horizontalLayout.addWidget(self.pushButton_2)
        self.retranslateUi(Form)
        QtCore.QMetaObject.connectSlotsByName(Form)
    def retranslateUi(self,Form):
        _translate=QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form","Form"))
        self.pushButton_2.setText(_translate("Form",""))

```

QPushButton、QlineEdit、QHBoxLayout、QWidget  
 Qt Designer、3-21

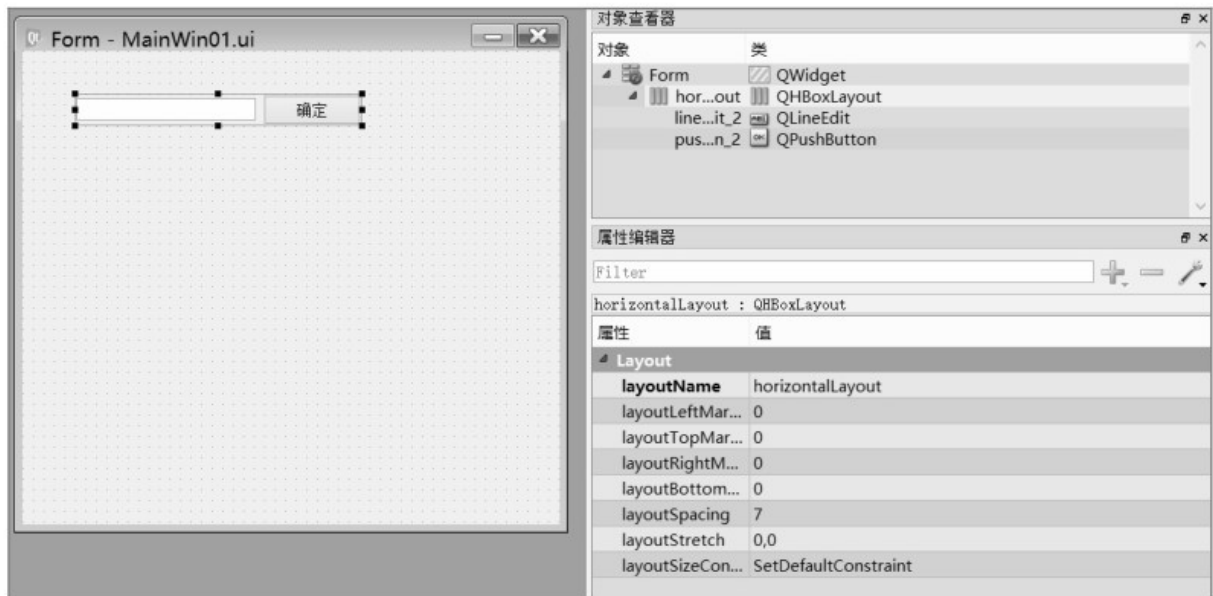


图3-21

0 \*Margin \* 0

MainWindow 3-22 PyQt5/Chapter03/layoutWin.ui



□3-22

11

## GridLayout “ ” 3

Python

3-23

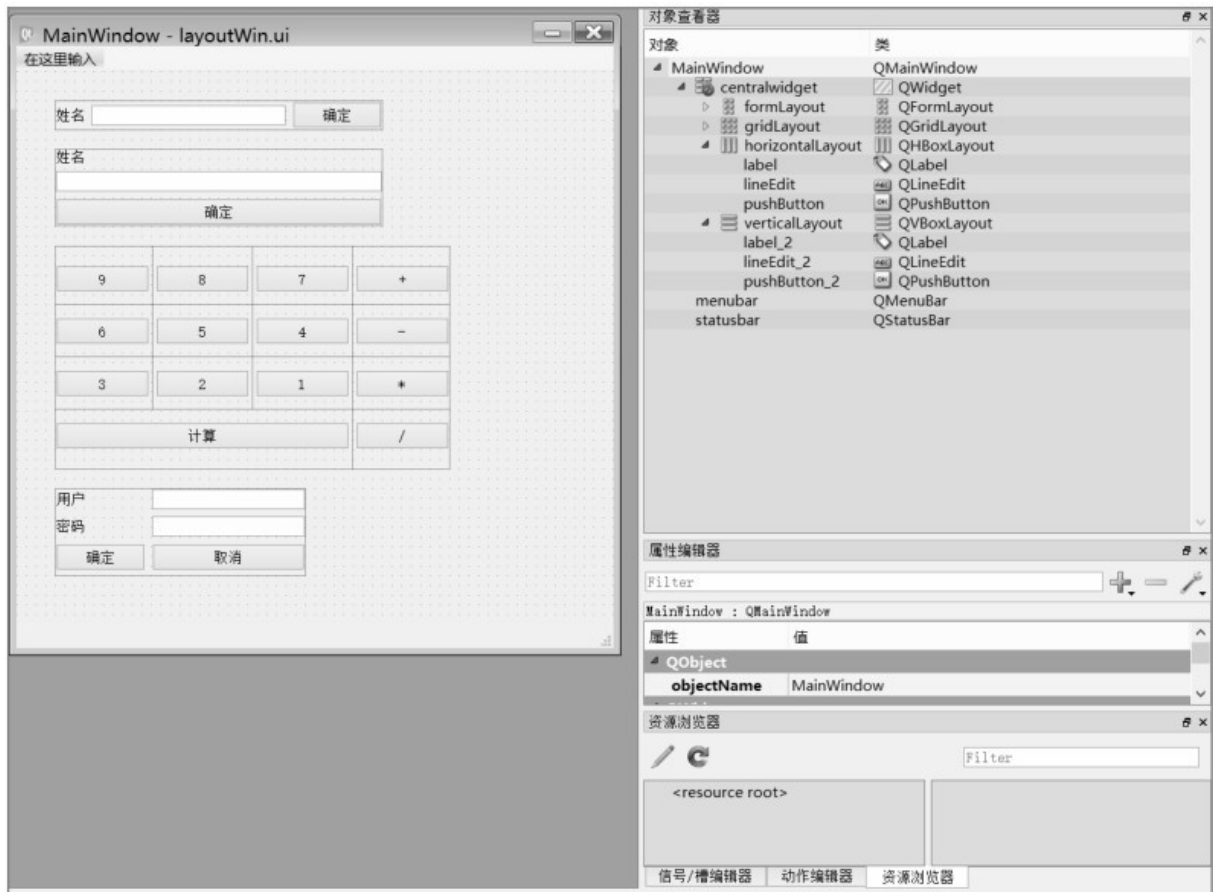


图 3-23

在 Qt Designer 中，我们创建了一个名为 MainWindow 的窗口，它包含了一个名为 layout 的布局。布局中包含了一个名为 pushButton 的按钮，一个名为 QLabel 的标签，以及一个名为 QLineEdit 的文本输入框。此外，还有一个名为 '计算' 的按钮，以及 '确定' 和 '取消' 按钮。用户可以在 '姓名' 和 '用户' 文本框中输入信息。

### 3.2.2 信号槽

在 Qt 中，信号槽机制是核心。信号 (Signal) 是由对象发出的，槽 (Slot) 是接收信号并执行特定操作的函数。在 MainWindow 窗口中，我们使用信号槽来连接 UI 元素。例如，当用户点击 '计算' 按钮时，会触发一个信号，该信号被连接到计算函数。同样，当用户点击 '确定' 按钮时，会触发一个信号，该信号被连接到确定函数。图 3-24 展示了信号槽的连接。

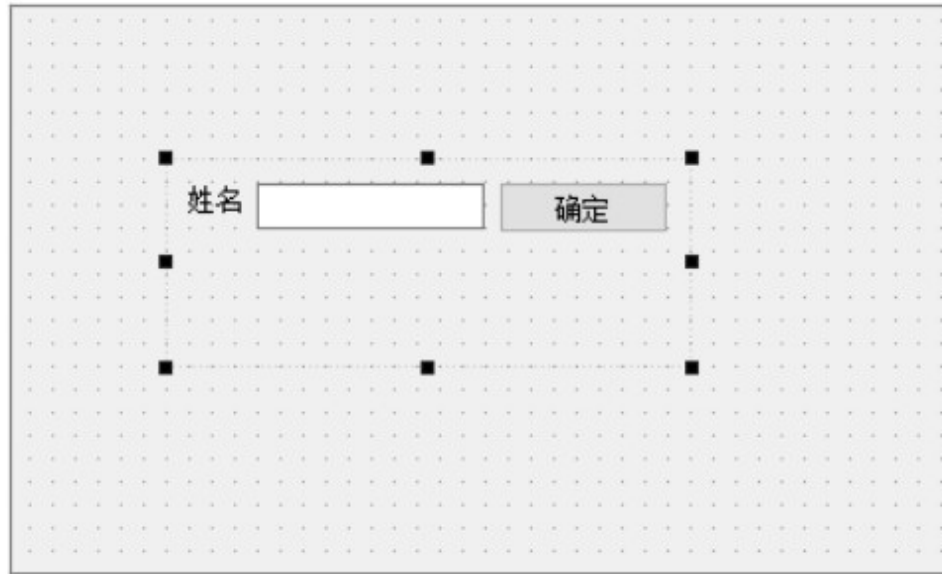


图3-24

将 Form 保存为名为“容器”的 Qt 文件 3-25

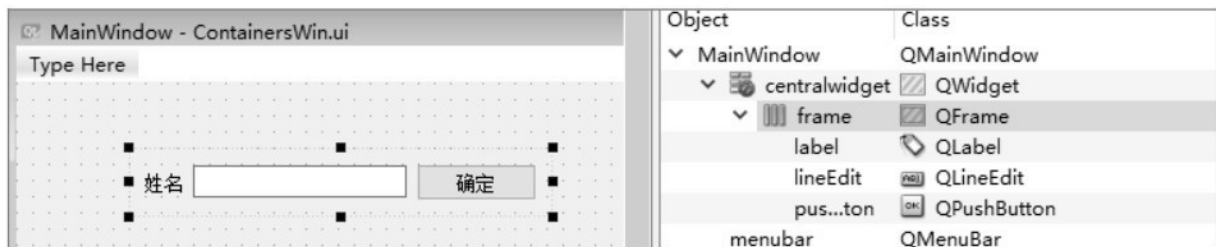


图3-25

将 PyQt5/Chapter03/ContainersWin.ui 文件  
ContainersWin.ui 保存为 ContainersWin.py 文件

```
#-*- coding: utf-8 -*-
```

```
from PyQt5 import QtCore, QtGui, QtWidgets
```

```
class Ui_MainWindow(object):
```

```
    def setupUi(self, MainWindow):
```

```
        MainWindow.setObjectName("MainWindow")
```

```
        MainWindow.resize(800,600)
```

```

self.centralwidget=QtWidgets.QWidget(MainWind
w)
self.centralwidget.setObjectName("centralwidget")
self.frame=QtWidgets.QFrame(self.centralwidget)
self.frame.setGeometry(QRect(70,40,264,4
3))
self.frame.setFrameShape(QtWidgets.QFrame.Style
dPanel)
self.frame.setFrameShadow(QtWidgets.QFrame.Rai
sed)
self.frame.setObjectName("frame")
self.horizontalLayout=QtWidgets.QHBoxLayout(self
.frame)
self.horizontalLayout.setObjectName("horizontalLa
yout")
self.label=QtWidgets.QLabel(self.frame)
self.label.setObjectName("label")
self.horizontalLayout.addWidget(self.label)
self.lineEdit=QtWidgets.QLineEdit(self.frame)
self.lineEdit.setObjectName("lineEdit")
self.horizontalLayout.addWidget(self.lineEdit)
self.pushButton=QtWidgets.QPushButton(self.fram
e)
self.pushButton.setObjectName("pushButton")
self.horizontalLayout.addWidget(self.pushButton)
MainWindow.setCentralWidget(self.centralwidget)
self.menubar=QtWidgets.QMenuBar(MainWindow)

```



```

        self.menubar.setGeometry(QRect(0,0,800,23))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar=QtWidgets.QStatusBar(MainWindow)

        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)
        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)
    def retranslateUi(self,MainWindow):
        _translate=QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow",
"MainWindow"))
        self.label.setText(_translate("MainWindow",""))
        self.pushButton.setText(_translate("MainWindow",""))
        self.QFrame.setObjectName("QHBoxLayout")

```

### 3.3 Qt Designer

Qt Designer is a graphical user interface (GUI) designer for Qt. It allows you to design the look and layout of your application's user interface without writing code. It is a part of the Qt framework and is used to create Qt-based applications. PyQt is a Python binding for Qt, and it allows you to use Qt's GUI components in Python. PyQt is a part of the Qt framework and is used to create Qt-based applications.

Qt Designer 的 Qt Designer 窗口中，单击 Main Window 窗口中的 Buttons 工具栏中的 Push Button 按钮，在画布上添加一个“按钮”对象。在对象的属性编辑器中，设置 geometry、sizePolicy、minimumSize、maximumSize 等属性。

图 3-26 显示了 Qt Designer 的属性编辑器窗口。在属性编辑器中，可以看到 QPushButton 对象的属性。其中，geometry 属性的值为 [(215, 140), 75 x 23]，表示按钮的几何信息。sizePolicy 属性的值为 [Minimum, Fixed, 0, 0]，表示按钮的大小策略。minimumSize 属性的值为 0 x 0，表示按钮的最小尺寸。maximumSize 属性的值为 16777215 x 16777215，表示按钮的最大尺寸。width 属性的值为 16777215，表示按钮的宽度。height 属性的值为 16777215，表示按钮的高度。sizeIncrement 属性的值为 0 x 0，表示按钮的尺寸增量。width 属性的值为 0，表示按钮的宽度。

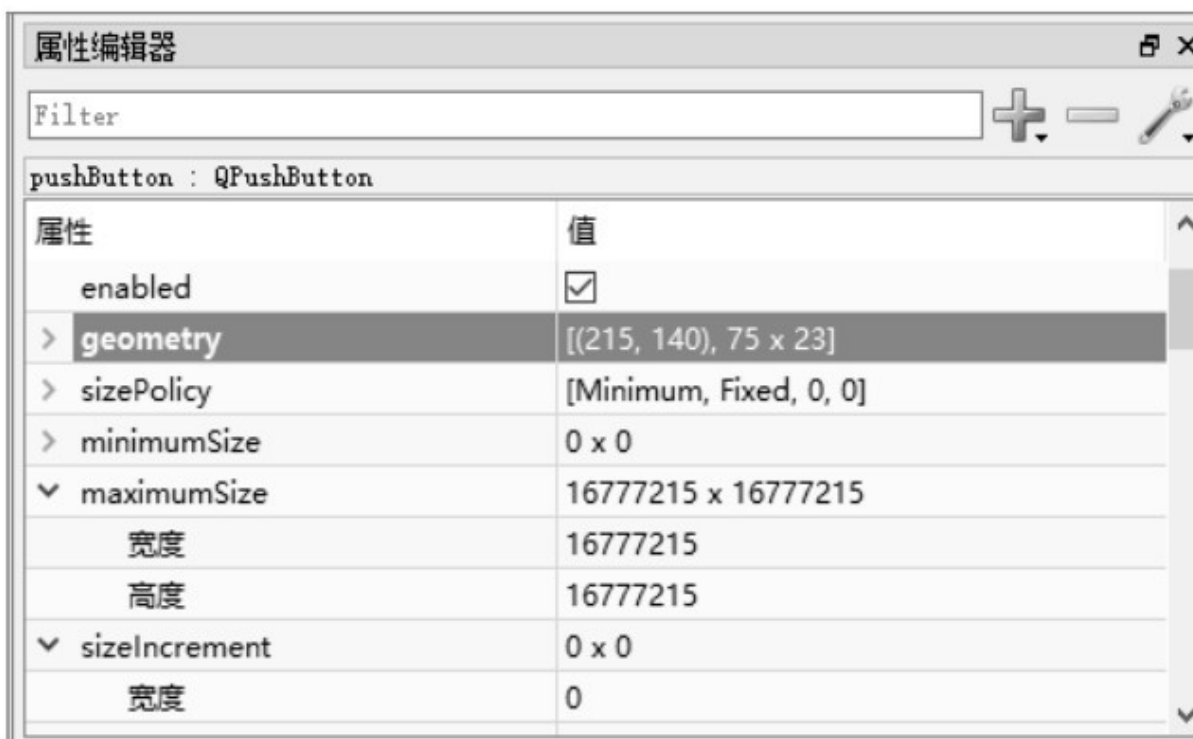


图 3-26

### 3.3.1 按钮

在 Qt Designer 中，单击 geometry 属性，可以打开 Geometry 窗口。在 Geometry 窗口中，可以设置按钮的几何信息。图 3-27 显示了 Geometry 窗口的截图。

▼ geometry	[(215, 140), 75 x 23]
X	215
Y	140
宽度	75
高度	23

图3-27

将按钮的几何信息设置为215px宽140px高，75px宽23px高。

```
self.pushButton=QtWidgets.QPushButton(self.centralwidget)
```

```
self.pushButton.setGeometry(QtCore.QRect(215,140,75,23))
```

```
self.pushButton.setObjectName("pushButton")
```

在代码清单3-27中，我们创建了一个QPushButton对象，并将其添加到主窗口中。

在代码清单3-27中，我们创建了一个QPushButton对象，并将其添加到主窗口中。

在代码清单3-27中，我们创建了一个QPushButton对象，并将其添加到主窗口中。

在代码清单3-27中，我们创建了一个QPushButton对象，并将其添加到主窗口中。





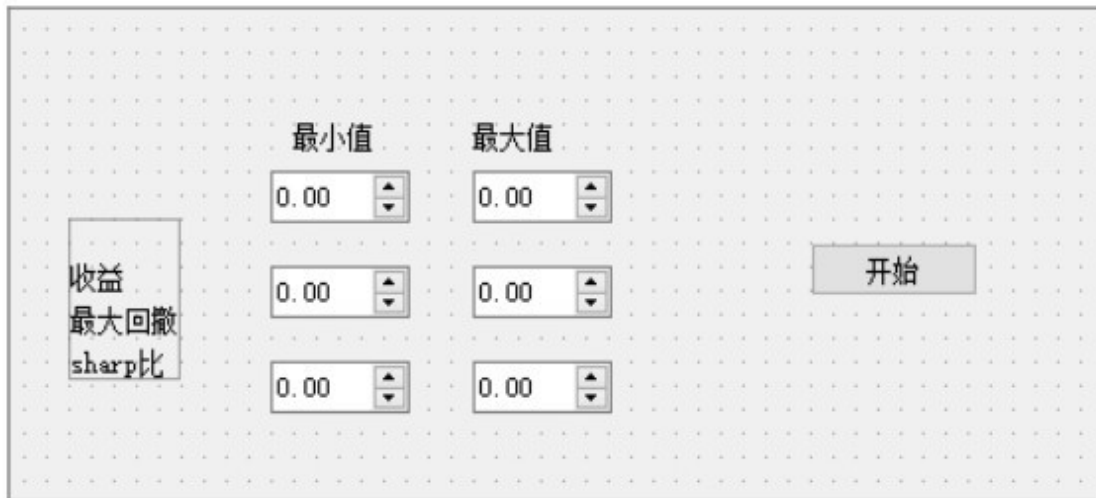


图3-30

```

self.verticalLayout.addWidget(self.label_6)
self.verticalLayout.addWidget(self.label_3)
self.verticalLayout.addWidget(self.label_4)
self.verticalLayout.addWidget(self.label_5)

```

```

self.verticalLayout.setObjectName("verticalLayout")
self.verticalLayout.addWidget(self.label_6)
self.verticalLayout.addWidget(self.label_3)
self.verticalLayout.addWidget(self.label_4)
self.verticalLayout.addWidget(self.label_5)

```

```

self

```

```

self.setObjectName("Label")
self.setObjectName("Label")

```

```

self.setObjectName("Label")
self.setObjectName("Label")

```

```

self.setObjectName("Label")
self.setObjectName("Label")

```

```

self.setObjectName("Qt Designer")
self.setObjectName("Qt Designer")

```

```

self.setGeometry(140,80,54,12)

```

```

self.setGeometry(140,80,54,12)

```

```

self

```

```

self.label.setGeometry(QRect(140,80,54,12))

```

```

self

```

## 2. 界面

界面如图 8 所示，包含 8 个文本框和 2 个按钮。其中“开始”按钮在右下角，其余 7 个文本框在左上角，排列如下：

最小值	最大值
0.00	0.00
0.00	0.00
0.00	0.00

收益最大回撤sharp比

开始

图3-31

界面如图 8 所示，包含 8 个文本框和 2 个按钮。其中“开始”按钮在右下角，其余 7 个文本框在左上角，排列如下：

```
self.gridLayout=QtWidgets.QGridLayout()
self.gridLayout.setObjectName("gridLayout")
self.gridLayout.addWidget(self.label,0,0,1,1)
# gridLayout.addWidget(文本框,行,列,列宽,行高)
# 文本框
self.gridLayout.addWidget(self.label_2,0,1,1,1)
self.gridLayout.addWidget(self.doubleSpinBox_returns_min,1,0,1,1)
self.gridLayout.addWidget(self.doubleSpinBox_returns_max,1,1,1,1)
self.gridLayout.addWidget(self.doubleSpinBox_maxdrawdown_min,2,0,1,1)
```

```

self.gridLayout.addWidget(self.doubleSpinBox_maxdrawdown_max,2,1,1,1)
self.gridLayout.addWidget(self.doubleSpinBox_sharp_min,3,0,1,1)
self.gridLayout.addWidget(self.doubleSpinBox_sharp_max,3,1,1,1)

```

### 3. 布局

1. Spacers 1.1 Horizontal Spacer 1.2 Vertical Spacer 1.3 Display Widgets 1.4 Horizontal Line 1.5 3-32

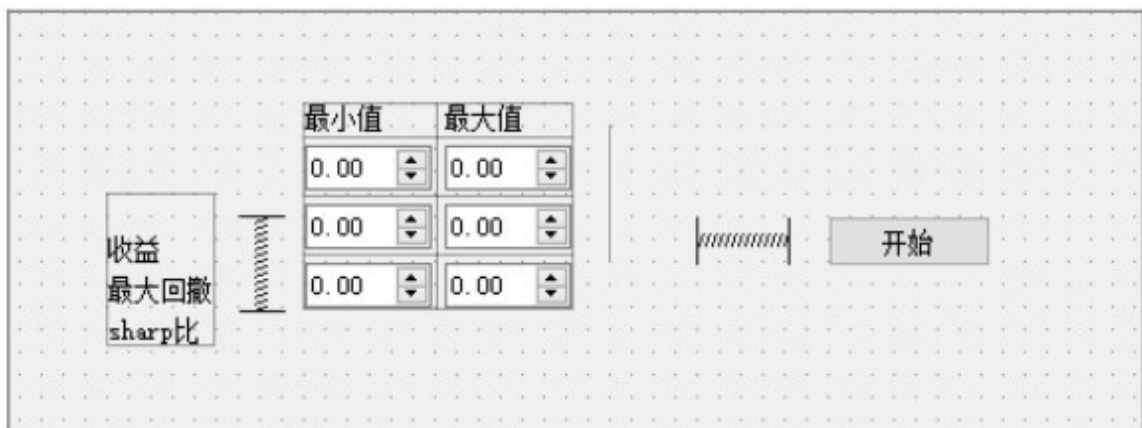


图3-32

- Vertical Spacer 1.1
- Horizontal Spacer 1.2
- Horizontal Line 1.3



```

self.line=QtWidgets.QFrame(self.widget)    # 画一条
Horizontal Line
self.line.setFrameShape(QtWidgets.QFrame.VLine)
self.line.setFrameShadow(QtWidgets.QFrame.Sunken)
self.line.setObjectName("line")
spacerItem1=QtWidgets.QSpacerItem(200,20,QtWidgets.
ts.QSizePolicy.Preferred,QtWidgets.QSizePolicy.Minimum) #
画Horizontal Spacer宽度200高度20大小策略为Preferred
spacerItem=QtWidgets.QSpacerItem(20,40,QtWidgets.
QSizePolicy.Minimum,QtWidgets.QSizePolicy.Expanding) #
画Vertical Spacer

```

将“开始”按钮移动到窗口的右下角，将“开始”按钮的
 3-33

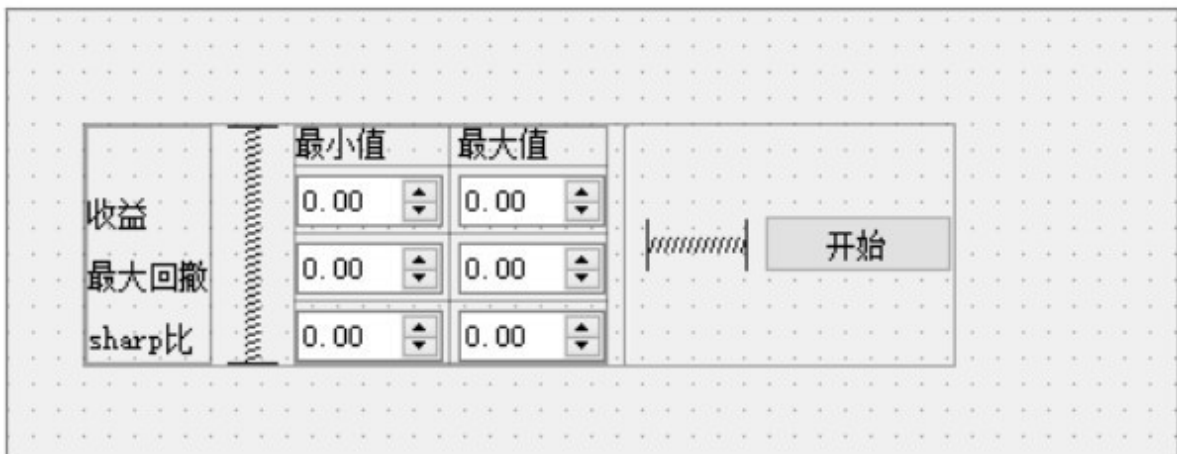
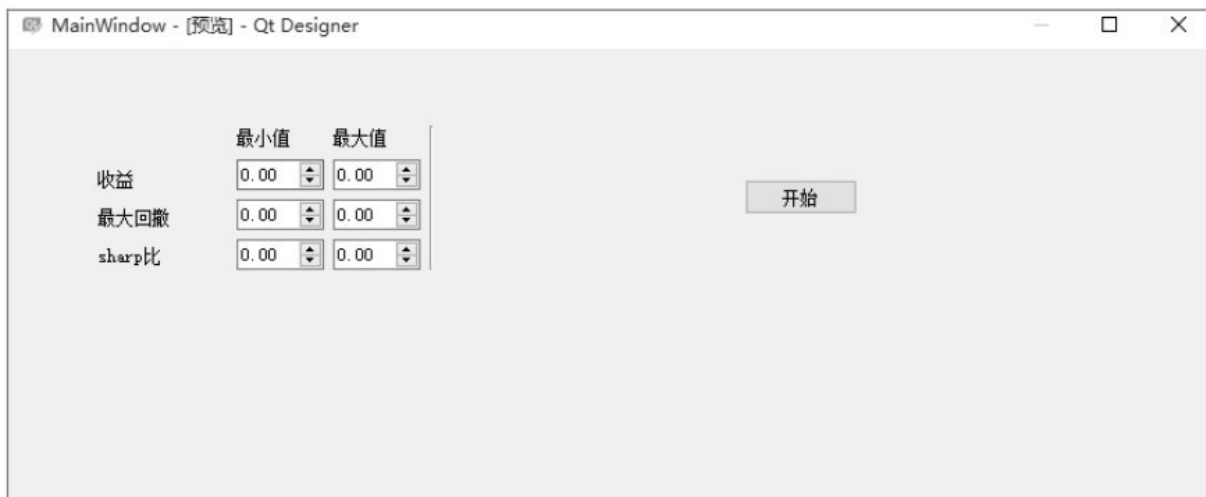


图3-33

将“开始”按钮移动到窗口的右下角，将“开始”按钮的
 horizontalSpacer宽度
 sizeType 为 preferred 高度 sizeHint 为 200 高度为 200
 horizontalSpacer宽度为 preferred 高度为 200
 200×200

□□□□□□□□□□“□□”→“□□”□□□□□3-34□□□



□3-34

[illegible]

```
class Ui_LayoutDemo(object): # 空类
LayoutDemo
    def setupUi(self,LayoutDemo):
LayoutDemo.setObjectName("LayoutDemo") # 设置窗口名称
LayoutDemo.resize(800,600)
        self.centralwidget=QtWidgets.QWidget(LayoutDem
o)
        # centralwidget
        self.centralwidget.setObjectName("centralwidget")
        self.layoutWidget=QtWidgets.QWidget(self.central
widget)
        # layoutWidget
        self.layoutWidget.setGeometry(QRect(90,90,391,161))
```

```

self.layoutWidget.setObjectName("layoutWidget")
self.horizontalLayout=QtWidgets.QHBoxLayout(self
.layoutWidget)
    # horizontalLayout QWidget
    self.horizontalLayout.setObjectName("horizontalLa
yout")
    # horizontalLayout QVBoxLayout
    self.horizontalLayout.addLayout(self.verticalLayout
)
    self.horizontalLayout.addItem(spacerItem)
    self.horizontalLayout.addLayout(self.gridLayout)
    self.horizontalLayout.addWidget(self.line)
    self.horizontalLayout.addItem(spacerItem1)
    self.horizontalLayout.addWidget(self.pushButton)

```

PyQt5 的 Qt Designer 是 Qt 的图形用户界面设计器，它允许用户通过拖放的方式设计 Qt 应用程序的图形用户界面。

在 Qt Designer 中，我们可以设置窗口的 `minimumSize`、`maximumSize` 和 `sizePolicy` 属性。

#### 4.minimumSize 和 maximumSize

`minimumSize` 和 `maximumSize` 属性用于设置窗口的最小和最大尺寸。

在 Qt Designer 中，我们可以设置窗口的 `minimumSize` 和 `maximumSize` 属性。

属性	值
▼ <b>minimumSize</b>	100 x 100
宽度	100
高度	100
▼ <b>maximumSize</b>	300 x 300
宽度	300
高度	300

图3-35

代码如下：

```
self.pushButton.setMinimumSize(QtCore.QSize(100,100))
self.pushButton.setMaximumSize(QtCore.QSize(300,300))
```

通过上述代码，我们设置好了按钮的大小。图3-36展示了运行后的效果。

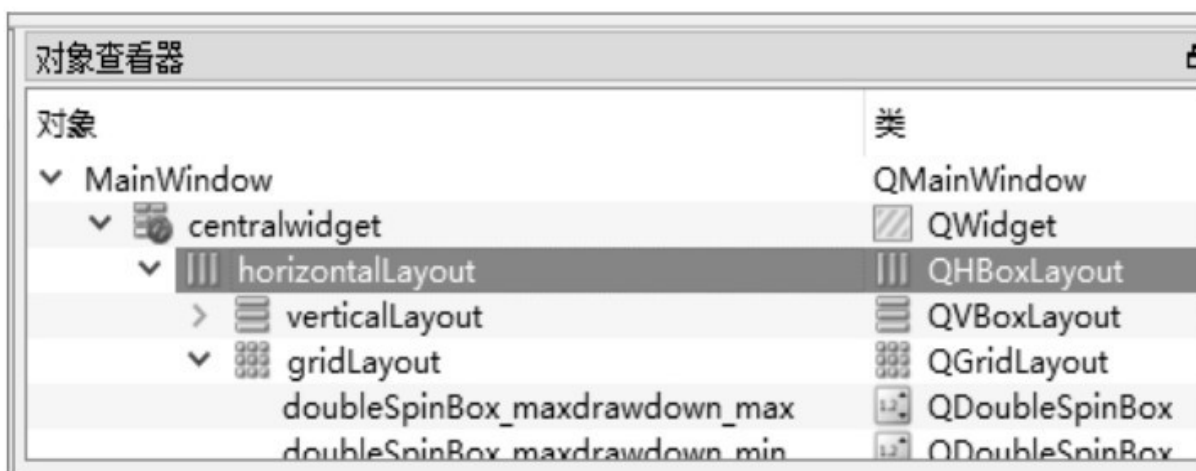
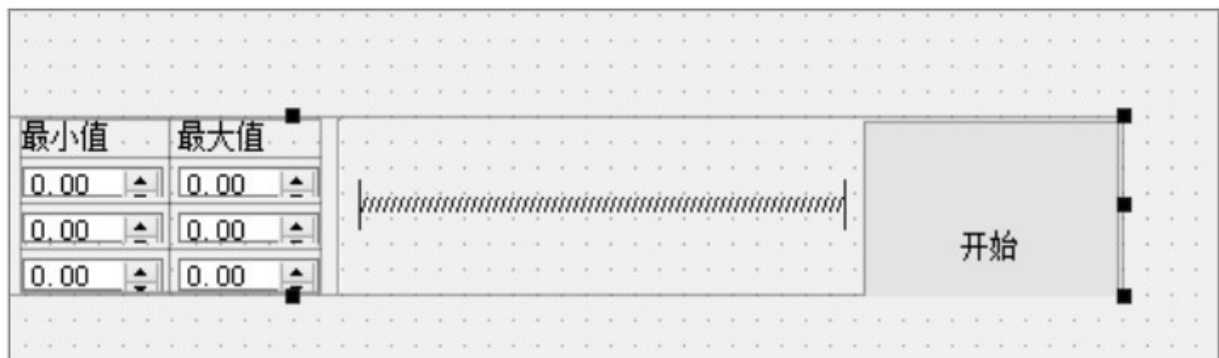


图3-36

[illegible]

□3-37

□□  
□□“□□”□□□□□□□ 100□□□□□□□□□□□□“□□”□□□□□□□□3-38□  
□□



pushButton : QPushButton	
Property	Value
▼ QObject	
objectName	pushButton
▼ QWidget	
enabled	<input checked="" type="checkbox"/>
> geometry	[(370, 130), 75 x 23]
> sizePolicy	[Minimum, Fixed, 0, 0]
> minimumSize	0 x 0

图3-39

## 5.sizePolicy

sizePolicy 和 sizeHint、minimumSize 一起使用

sizeHint 和 minimumSize 一起使用

minimumSize 和 sizePolicy 一起使用

sizePolicy 和 sizeHint 一起使用

sizePolicy 和 sizeHint 一起使用

sizePolicy 和 sizeHint 一起使用

图3-40 大小策略 sizePolicy

✓ sizePolicy	[Minimum, Fixed, 0, 0]
水平策略	Minimum
垂直策略	Fixed
水平伸展	0
垂直伸展	0

图3-40

大小策略

```

sizePolicy=QtWidgets.QSizePolicy(QtWidgets.QSizePolicy.Fixed,QtWidgets.QSizePolicy.Minimum)
sizePolicy.setHorizontalStretch(0) # 0
sizePolicy.setVerticalStretch(0) # 0
sizePolicy.setHeightForWidth(self.pushButton.sizePolicy().hasHeightForWidth())
self.pushButton.setSizePolicy(sizePolicy)

```

QtWidgets.QSizePolicy

● Fixed 0 0 sizeHint 0 0

● Minimum 0 0 sizeHint 0 0

● Maximum 0 0 sizeHint 0 0

● Preferred 0 0 sizeHint 0 0

● Expanding 0 0 minsizeHint 0 0 sizeHint 0 0

● MinimumExpanding 0 0 sizeHint 0 0

● Ignored 0 0 sizeHint 0 0 minsizeHint 0 0

Minimum 0 0 sizeHint 0 0

Maximum 0 0 sizeHint 0 0

Minimum Maximum 0 0

0 0

0 0 “0” “0” “sharp” 1 3 1

horizontalLayout 3-41



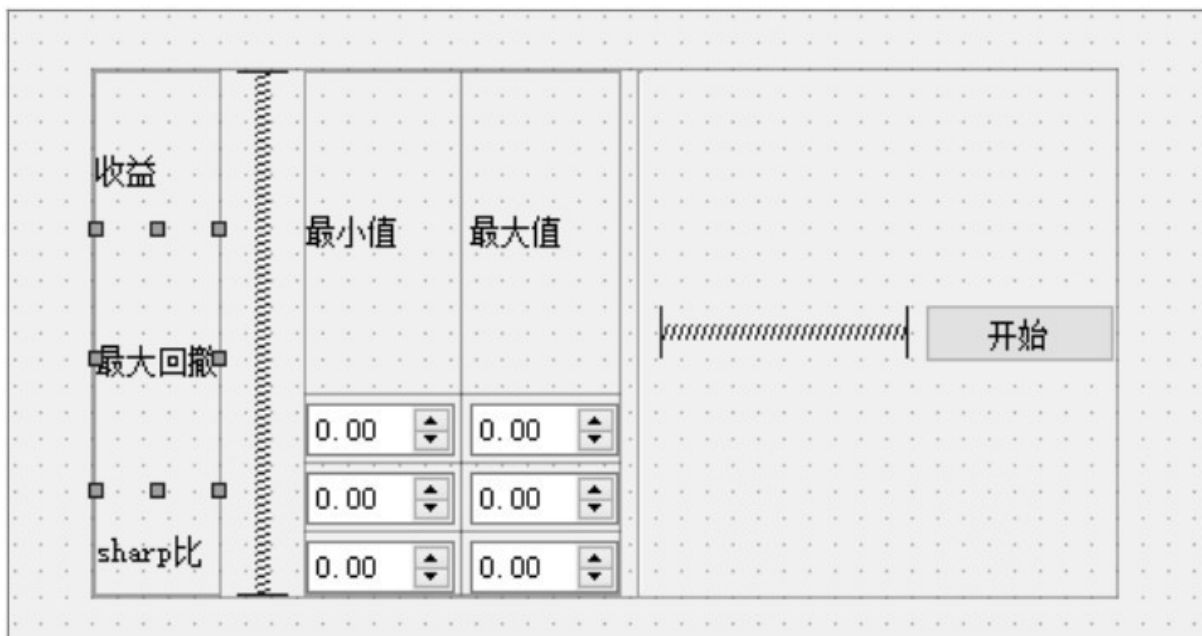


图3-41

在代码中，我们使用“QtWidgets.QSizePolicy”来设置窗口的策略，使其在水平方向上拉伸，而在垂直方向上拉伸。代码如下：

```
sizePolicy = QtWidgets.QSizePolicy(
    QtWidgets.QSizePolicy.Preferred, QtWidgets.QSizePolicy.Preferred)
sizePolicy.setHorizontalStretch(0)
sizePolicy.setVerticalStretch(1)
```



□4□□□□□□□□□□□□□□ Vertical Spacer□Horizontal Spacer□  
 Horizontal Line□Vertical Line□□□□□□□□□□□□□□□□□□□□

5 splitter

[illegible]

**7**

[illegible]

9 Tab

10

**011**

12

13 Eric pyuic5  
Eric 3.4.2 “ ”

[illegible]

□□□□□□1□6□11□14□□□□□□□□□□□□7□10□□□□□□□□□□□□  
7□9□□□□□□□10□

## 2. □□□□□□□□□□□□

[illegible][illegible]

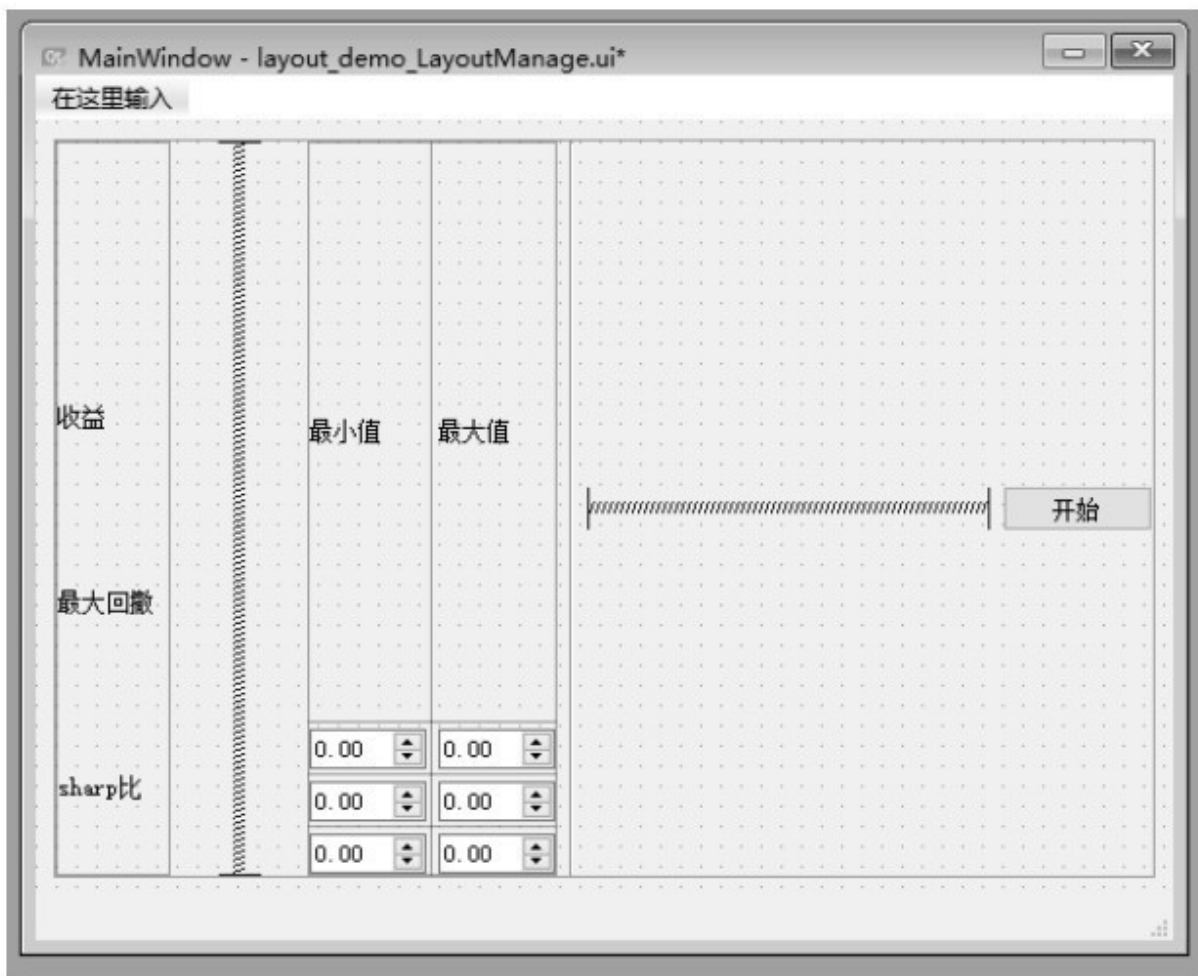


图3-42

1. 在MainWindow.ui文件中，找到以下代码：
   
 2. 将以下代码替换为以下代码：
   
 3. 编译并运行程序，可以看到MainWindow.ui文件中的布局已经按照要求进行了修改。

### 3. 修改布局

1. 在MainWindow.ui文件中，找到以下代码：
   
 2. 将以下代码替换为以下代码：
   
 3. 编译并运行程序，可以看到MainWindow.ui文件中的布局已经按照要求进行了修改。



图3-43

代码如下：

```
self.label_5.setBuddy(self.doubleSpinBox_sharp_min)
```

“”→“”“Ctrl+R”

“Alt+S”

“doubleSpinBox\_sharp\_min” label doubleSpinBox

Display Widgets Input Widgets

Display Widgets

“”“”

#### 4. Tab

“Edit”→“Tab”3-44

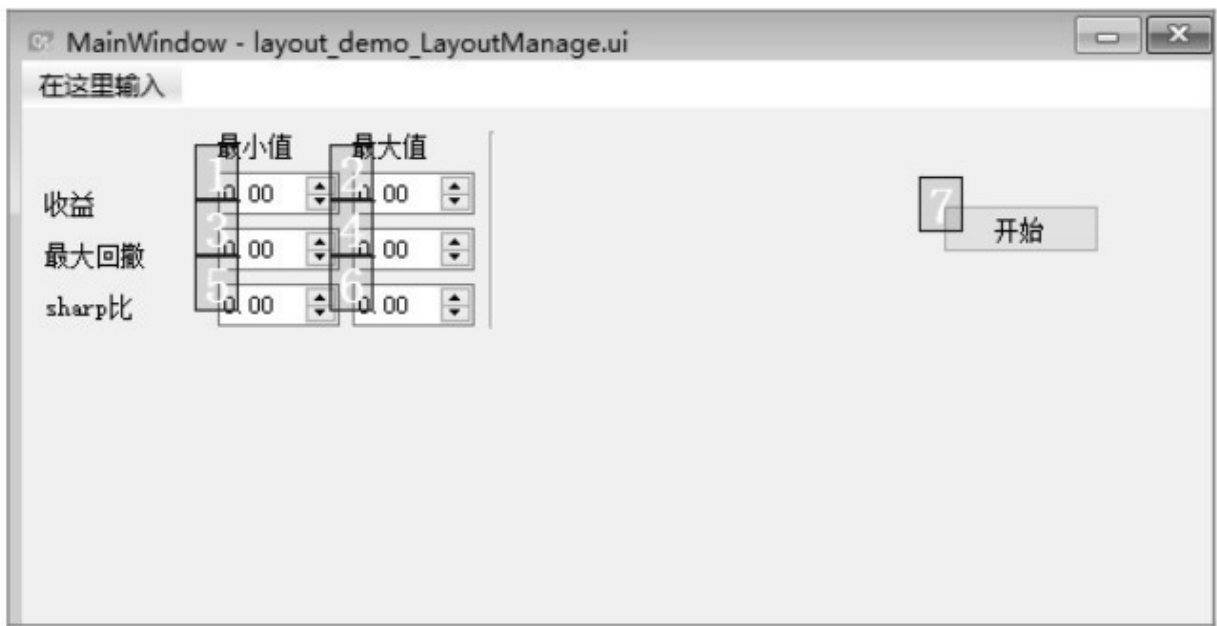


图3-44

图3-44中的1~7分别代表Tab页的标题，其中Tab页的标题为“开始”。

在图3-44中，Tab页的标题为“开始”，其中“开始”为Tab页的标题。

在图3-44中，Tab页的标题为“开始”，其中“开始”为Tab页的标题。

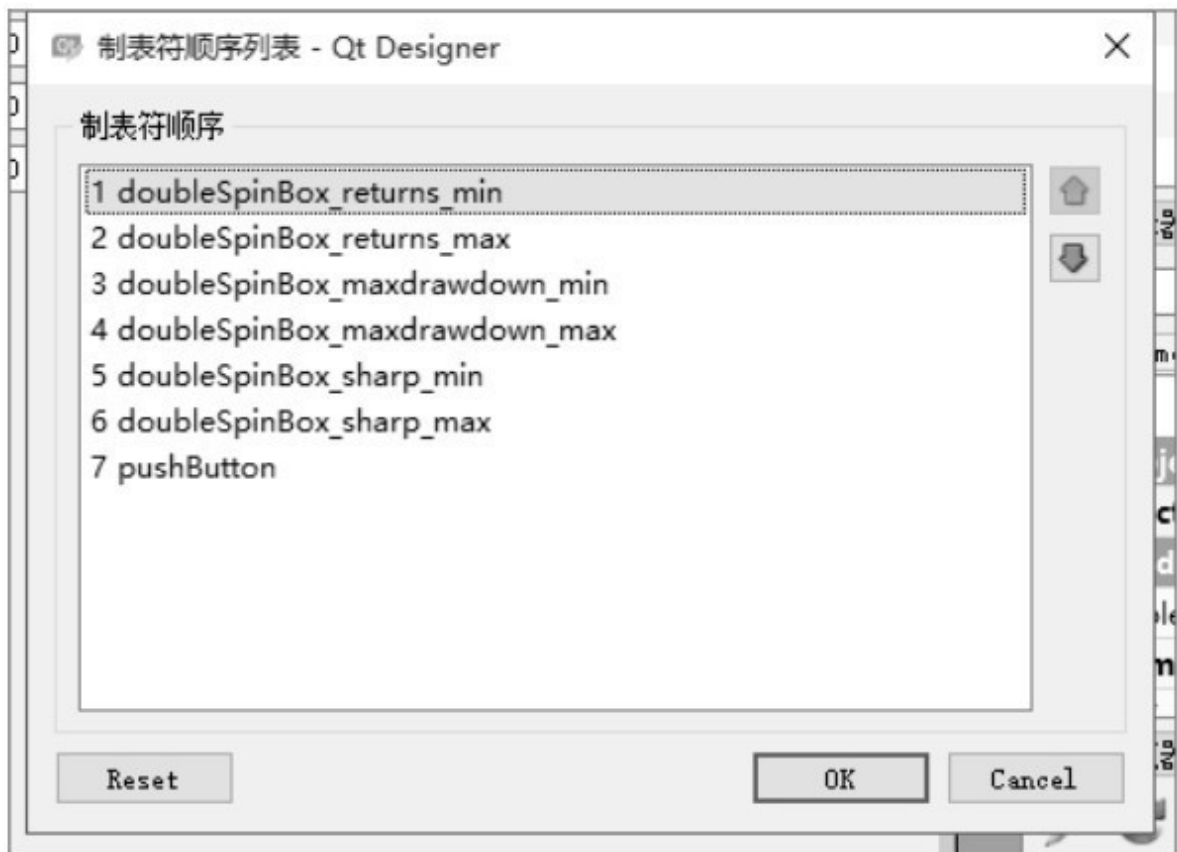


图3-45

在 Qt Designer 中，可以通过“制表符顺序列表”对话框来设置控件的制表符顺序。该对话框显示了当前设计中的控件及其制表符顺序。您可以通过单击“Reset”按钮来重置制表符顺序，或者通过单击“OK”按钮来保存更改。

### 3.3.4 布局管理

在 3.1.4 节中，我们介绍了如何使用 Qt Designer 来设计用户界面。在 Eric 6 中，我们可以使用布局管理器来管理控件的布局。布局管理器可以帮助我们轻松地创建复杂的用户界面，并确保控件在窗口中正确地排列。

```

# -*- coding: utf-8 -*-

"""
Module implementing LayoutDemo.
"""

from PyQt5.QtCore import pyqtSlot
from PyQt5.QtWidgets import QMainWindow, QApplication
from Ui_layout_demo_LayoutManage import Ui_LayoutDemo

class LayoutDemo(QMainWindow, Ui_LayoutDemo):
    """
    Class documentation goes here.
    """
    def __init__(self, parent=None):
        """
        Constructor

        @param parent reference to the parent widget
        @type QWidget
        """
        super(LayoutDemo, self).__init__(parent)
        self.setupUi(self)

    @pyqtSlot()
    def on_pushButton_clicked(self):
        """
        Slot documentation goes here.
        """
        print('收益_min:', self.doubleSpinBox_returns_min.text())
        print('收益_max:', self.doubleSpinBox_returns_max.text())
        print('最大回撤_min:', self.doubleSpinBox_maxdrawdown_min.
text())
        print('最大回撤_max:', self.doubleSpinBox_maxdrawdown_max.
text())
        print('sharp 比_min:', self.doubleSpinBox_sharp_min.text())
        print('sharp 比_max:', self.doubleSpinBox_sharp_max.text())

if __name__ == "__main__":

```





```
2 >>> 收益_min: 0.30
3 收益_max: 0.70
4 最大回撤_min: 0.50
5 最大回撤_max: 1.00
6 sharp比_min: 3.23
7 sharp比_max: 5.64
8
```

图3-47

## 3.4 信号槽

信号槽(signal-slot)是Qt中实现对象间通信的机制。在PyQt 5中，可以通过QObject.signal.connect()方法将信号与槽函数连接起来。

QObject是Qt中的基类，QWidget是其派生类。在PyQt 5中，QWidget是QtWidgets模块的一部分。通过QWidget，我们可以创建各种窗口和控件。

Qt信号槽机制允许QObject对象发出信号，而其他QObject对象则可以连接到这些信号并执行相应的槽函数。这种机制使得Qt应用程序的组件之间可以灵活地进行通信。

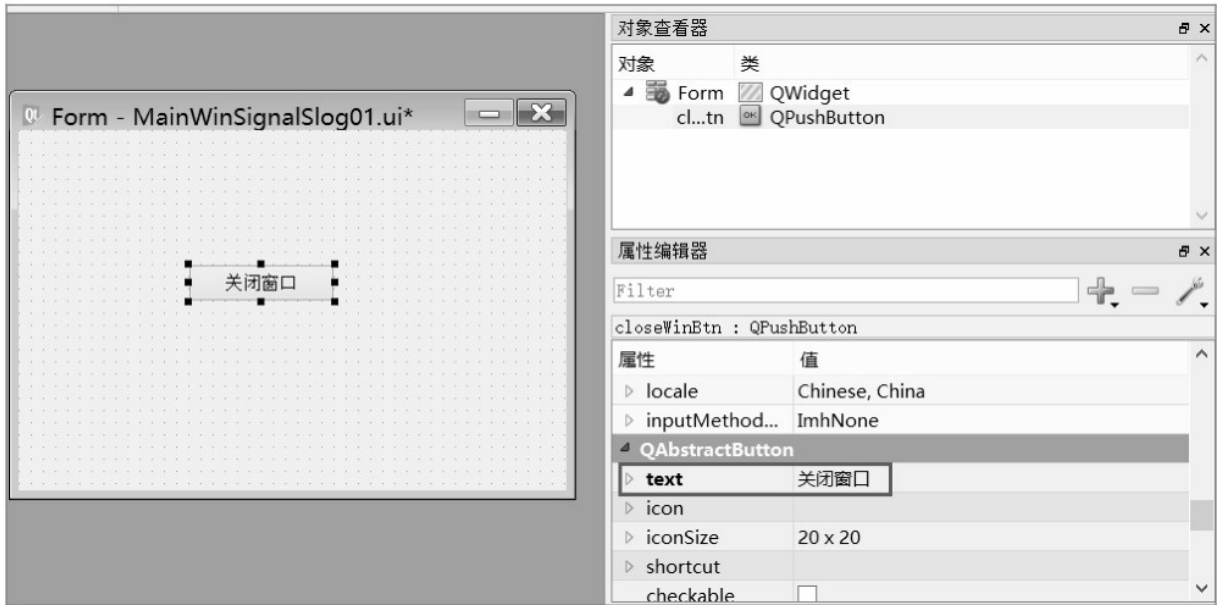
在PyQt 5中，我们可以使用Eric IDE来创建和运行Qt应用程序。Eric提供了一个图形用户界面，使得开发Qt应用程序变得更加简单。

### 3.4.1 Qt Designer

Qt Designer是Qt开发工具包中的一个组件，用于创建Qt应用程序的图形用户界面。它提供了一个可视化的设计器，允许用户通过拖放控件来设计界面。设计好的界面可以保存为.ui文件，然后在Python代码中加载和使用。例如，MainWinSignalSlog01.ui就是一个使用Qt Designer设计的界面文件。

PyQt5/Chapter03/MainWinSignalSlog01.ui

Qt DesignerButtonsQPushButtonForm text “” objectName“closeWinBtn”3-48



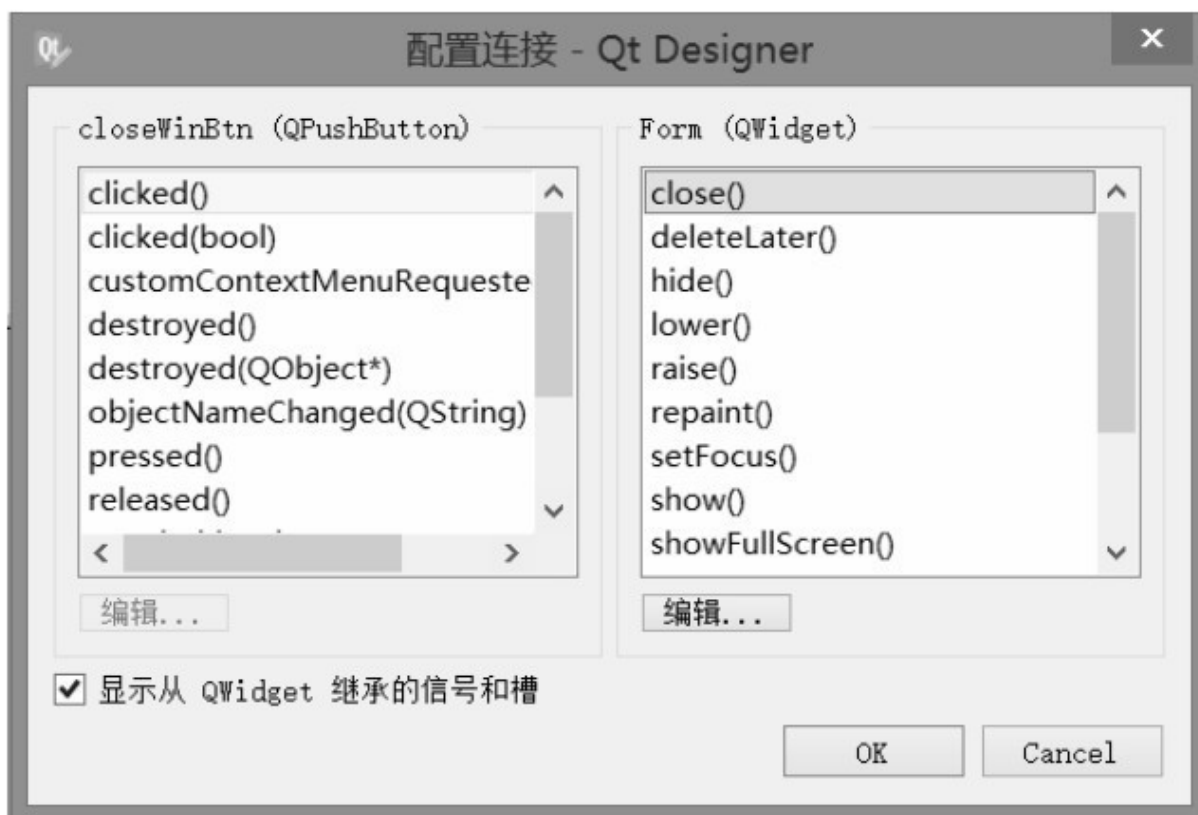
3-48

“/”“Edit”→“/” “”Form 3-49



图3-49

“关闭窗口”按钮的信号槽



□3-50

[illegible]

```

        QMessageBox::Close = QMessageBox::Close;
        closeWinBtn.clicked().connect(Form, &Form::close);
    }
};

```

“ ” 3-51

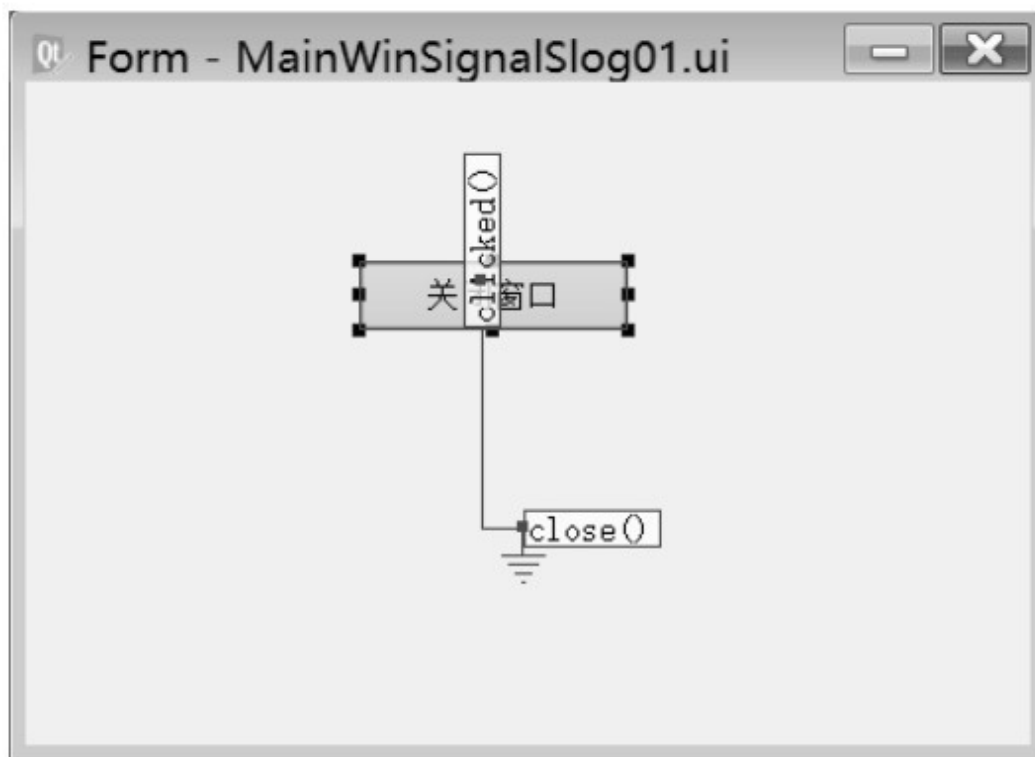


图3-51

下面将使用 Python 语言编写主程序 MainWinSignalSlog01.ui  
 并保存为 MainWinSignalSlog01.py 文件，并编译生成  
 MainWinSignalSlog01.ui 文件，编译生成 .py 文件  
 pyuic5-o MainWinSignalSlog01.py  
 MainWinSignalSlog01.ui

下面将编写 MainWinSignalSlog01.py 文件

```
from PyQt5 import QtCore,QtGui,QtWidgets
```

```
class Ui_Form(object):
```

```
    def setupUi(self,Form):
```

```
        Form.setObjectName("Form")
```

```
        Form.resize(452,296)
```

```
        self.closeWinBtn=QtWidgets.QPushButton(Form)
```

```

        self.closeWinBtn.setGeometry(QtCore.QRect(150,8
0,121,31))
        self.closeWinBtn.setObjectName("closeWinBtn")
        self.retranslateUi(Form)
        self.closeWinBtn.clicked.connect(Form.close)
        QtCore.QMetaObject.connectSlotsByName(Form)
    def retranslateUi(self,Form):
        _translate=QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form","Form"))
        self.closeWinBtn.setText(_translate("Form","关闭"))
        pyuic5 PyQt5 Python  PyQt5  closeWinBtn
clicked()  Form.close()
QObject.signal.connect()
        self.closeWinBtn.clicked.connect(Form.close)
    pyuic5  QtCore.QMetaObject.connect
SlotsByName(Form)
    7
    close()
    CallMainWinSignalSlog01.py
import sys
from PyQt5.QtWidgets import QApplication
,QMainWindow
from MainWinSignalSlog01 import Ui_Form
class MyMainWindow(QMainWindow,Ui_Form):

```

```

def __init__(self,parent=None):
    super(MyMainWindow,self).__init__(parent)
    self.setupUi(self)
if __name__=="__main__":
    app=QApplication(sys.argv)
    myWin=MyMainWindow()
    myWin.show()
    sys.exit(app.exec_())

```

图3-52

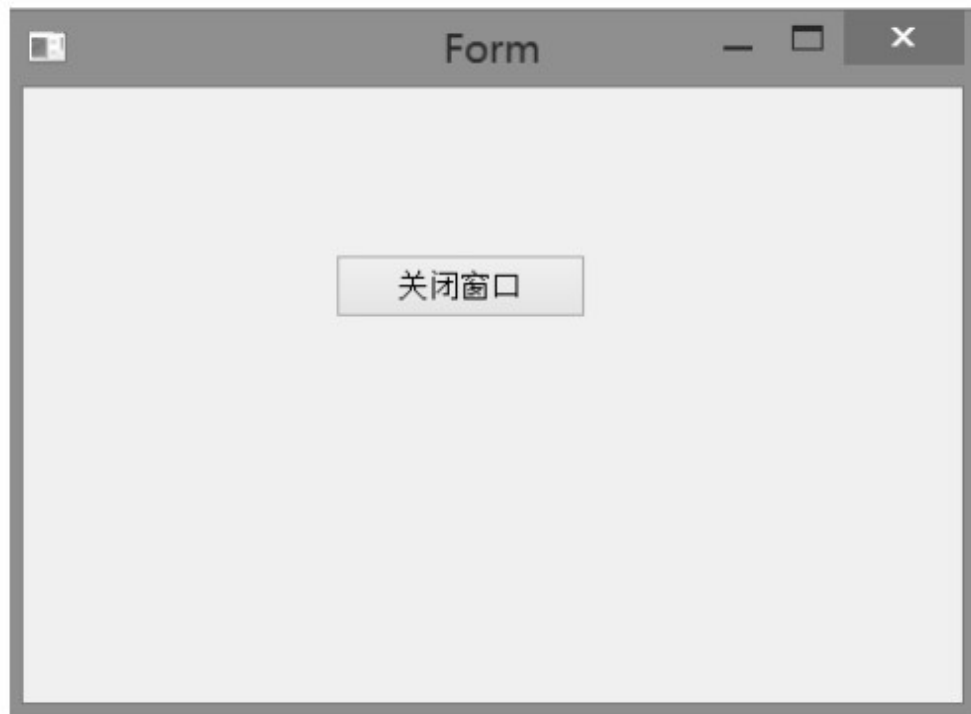


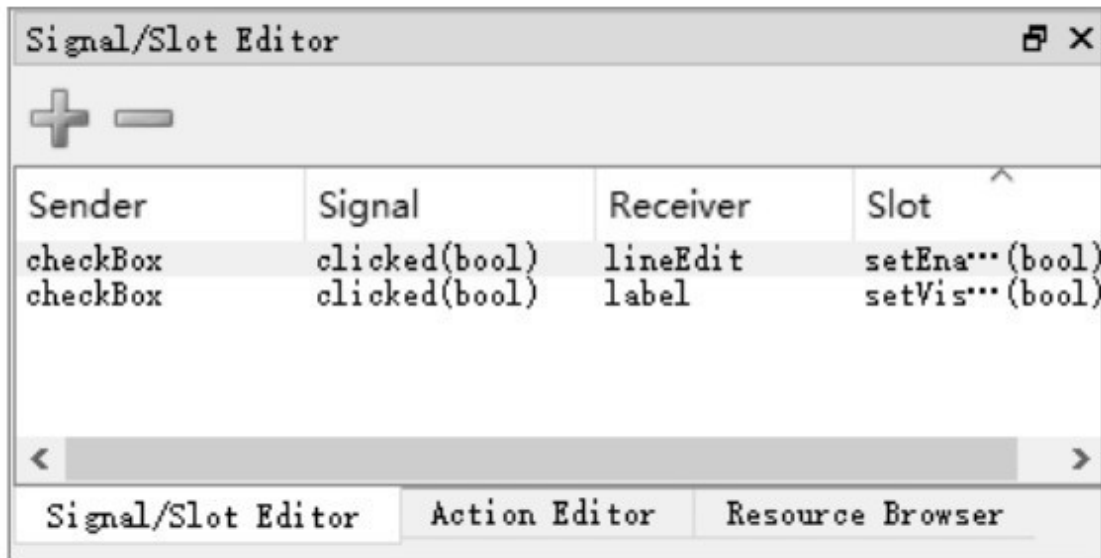
图3-52

### 3.4.2

PyQt



3-50
   
 3-53



3-53

MainWinSignalSlog03.ui
   
 MainWinSignalSlog03.py
   
 CallMainWinSignalSlog03.py

MainWinSignalSlog03.ui

MainWinSignalSlog03.py

CallMainWinSignalSlog03.py

CallMainWinSignalSlog03.py

3-54



图3-54

图3-55“显示”对话框

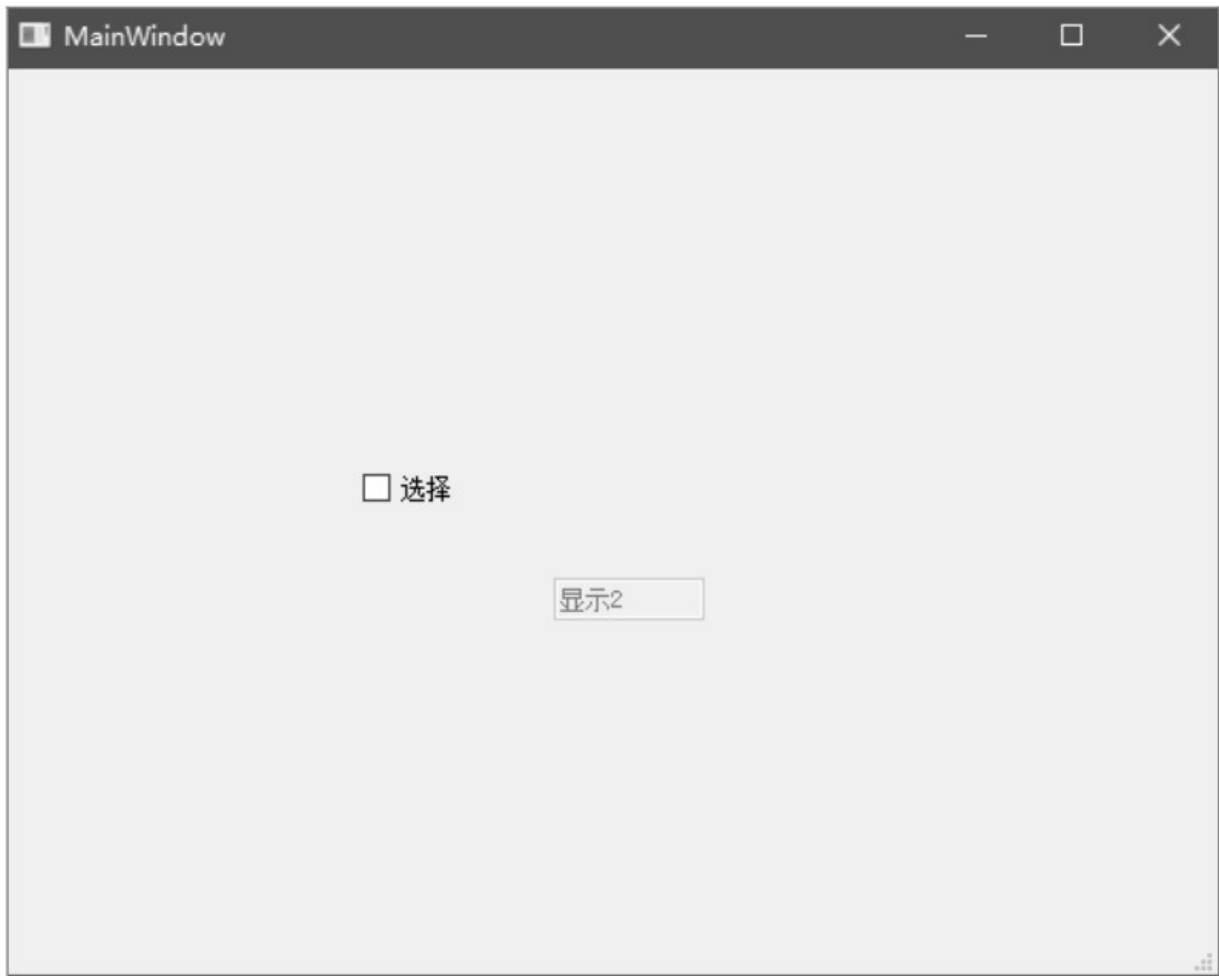


图3-55

```

        checkBox.clicked.connect(lambda: self.label.setVisible(
            lineEdit.setEnabled(True)))
        self.checkBox.clicked['bool'].connect(self.label.setVisible)
        self.checkBox.clicked['bool'].connect(self.lineEdit.setEnabled)

        # 生成UI文件
        ui = QtWidgets.QMainWindow()
        ui.setWindowTitle('MainWindow')
        ui.setObjectName('MainWindow')
        ui.resize(400, 300)
        ui.setStyleSheet('background-color: #f0f0f0;')
        ui.show()
        ui.setWindowTitle('MainWindow')
        ui.setObjectName('MainWindow')
        ui.resize(400, 300)
        ui.setStyleSheet('background-color: #f0f0f0;')
        ui.show()

```

□MainWinSignalSlog03.ui□□□MainWinSignalSlog04.ui□□□□  
□□□□□□□□□□□□□□□□

□□□Eric□“□□”□□□□□□MainWinSignalSlog04.ui□□□□□□□□□□  
“□□□□”□□□□Ui\_MainWinSignalSlog04.py□□□□□□□□□□□□□□□□“□□□□□□  
□□”□□□3-56□□□□



□3-56

在Eric中打开“文件”菜单，选择“新建”

图3-57

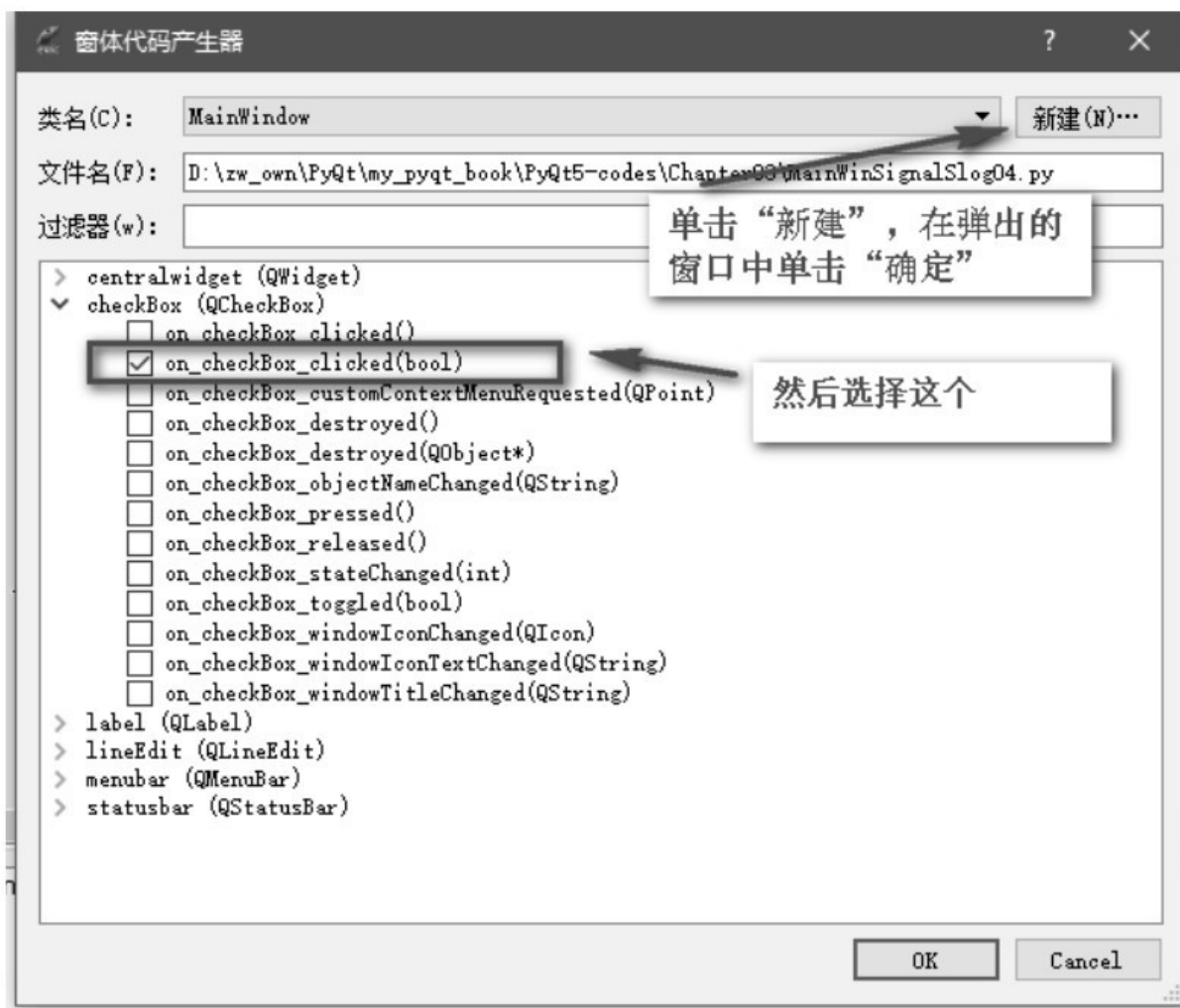


图3-57

单击“OK”按钮，生成代码并保存为MainWinSignalSlog04.py

```

# -*- coding: utf-8 -*-

"""
Module implementing MainWindow.
"""

from PyQt5.QtCore import pyqtSlot
from PyQt5.QtWidgets import QMainWindow

from Ui_MainWinSignalSlog04 import Ui_MainWindow

#注：原代码为 from .Ui_MainWinSignalSlog04 import Ui_MainWindow, 运行出
错，需要去掉

class MainWindow(QMainWindow, Ui_MainWindow):
    """
    Class documentation goes here.
    """
    def __init__(self, parent=None):
        """
        Constructor

        @param parent reference to the parent widget
        @type QWidget
        """
        super(MainWindow, self).__init__(parent)
        self.setupUi(self)

    @pyqtSlot(bool)
    def on_checkBox_clicked(self, checked):
        """
        Slot documentation goes here.

        @param checked DESCRIPTION
        @type bool
        """
        # TODO: not implemented yet
        raise NotImplementedError

```

MainWinSignalSlog04.py

```
# -*- coding: utf-8 -*-
```

```
"""
```

```
Module implementing MainWindow.
```

```
"""
```

```
from PyQt5.QtCore import pyqtSlot
```

```
from PyQt5.QtWidgets import QMainWindow, QApplication
```

```
from Ui_MainWinSignalSlog04 import Ui_MainWindow
```

#注: 原代码为 from .Ui\_MainWinSignalSlog04 import Ui\_MainWindow, 运行出错, 需要去掉

```
class MainWindow(QMainWindow, Ui_MainWindow):
```

```
    """
```

```
    Class documentation goes here.
```

```
    """
```

```
    def __init__(self, parent=None):
```

```
        """
```

```
        Constructor
```

```
        @param parent reference to the parent widget
```

```
        @type QWidget
```

```
        """
```

```
        super(MainWindow, self).__init__(parent)
```

```
        self.setupUi(self)
```

```
        self.checkBox.setChecked(True) # 设置 checkBox 默认的初始状态为选择
```

```
    @pyqtSlot(bool)
```

```
    def on_checkBox_clicked(self, checked):
```

```
        """
```

```
        Slot documentation goes here.
```

```
        @param checked DESCRIPTION
```

```
        @type bool
```

```
        """
```

```
        self.label.setVisible(checked)
```

```
        self.lineEdit.setEnabled(checked)
```

```
if __name__ == "__main__":
```



```
import sys
app = QApplication(sys.argv)
myWin = MainWindow()
myWin.show()
sys.exit(app.exec_())
```

                3-55                ——                

```
PyQt5.QtWidgets.QWidget.__init__
PyQt5.QtCore.QObject.__init__
```

PyQt7

### 3.5 □□□□□□□

### 3.5.1 背景

MainWindow 00000000000000000000000000000000“00000”  
000“00(&F)”00“00(&E)”00  
0000000000003-580000000000000000000000

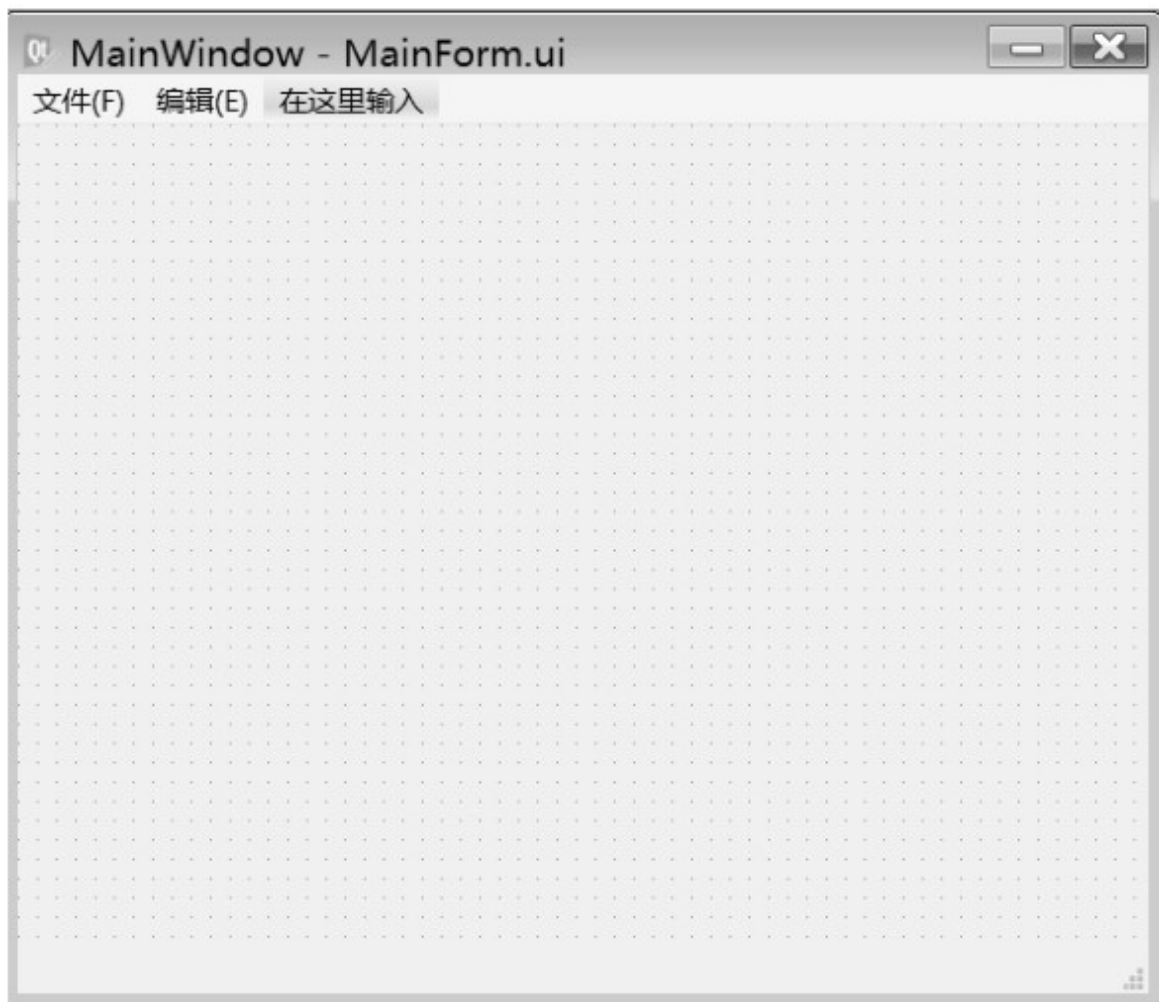


图3-58

在Qt Designer中，将“文件”→“编辑”菜单项替换为“Ctrl+R”菜单项，如图3-59所示。

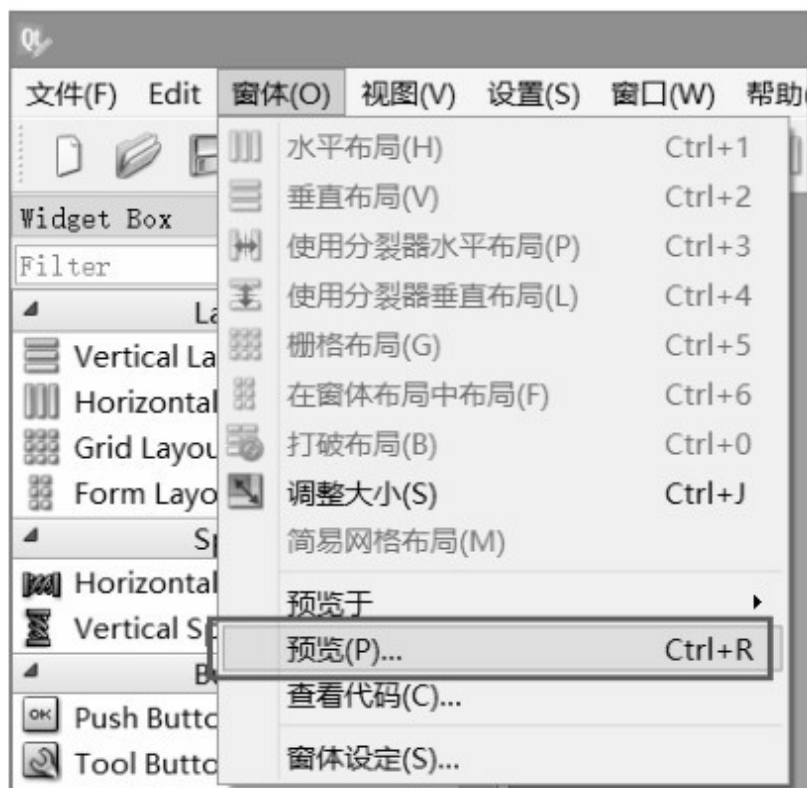


图3-59

在“窗口”菜单中，选择“预览(P)...”选项，即可在运行时预览窗体的外观。图3-60所示为“动作编辑器”窗口，其中列出了窗体的各种动作，如“打开”、“新建”、“关闭”和“添加窗体”等。每个动作都有一个复选框，用于启用或禁用该动作。此外，每个动作还显示了其对应的快捷键、是否可选以及工具提示。



图3-60

在“动作编辑器”窗口中，可以通过勾选“使用”列中的复选框来启用或禁用特定的动作。图3-61所示为“信号/槽编辑器”窗口，其中列出了窗体的各种信号和槽函数，以及它们之间的连接关系。



图3-61

在Qt Designer中，可以通过“编辑动作”对话框来配置动作的属性。图3-62显示了“编辑动作”对话框的“动作”选项卡。

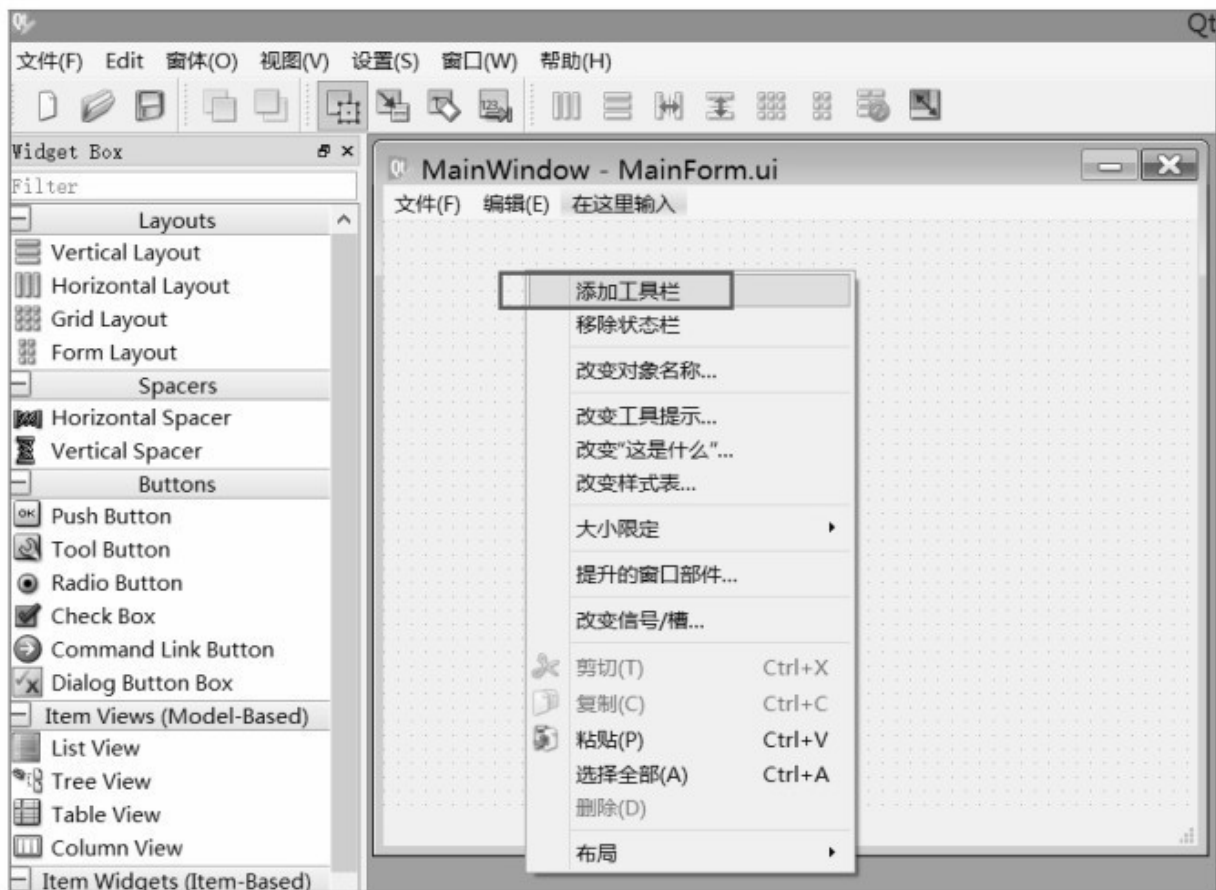


图3-62

Qt Designer 添加 WinAction 3-63



图3-63

在 Qt Designer 中，可以通过“编辑动作”对话框来配置动作的属性。

图3-64

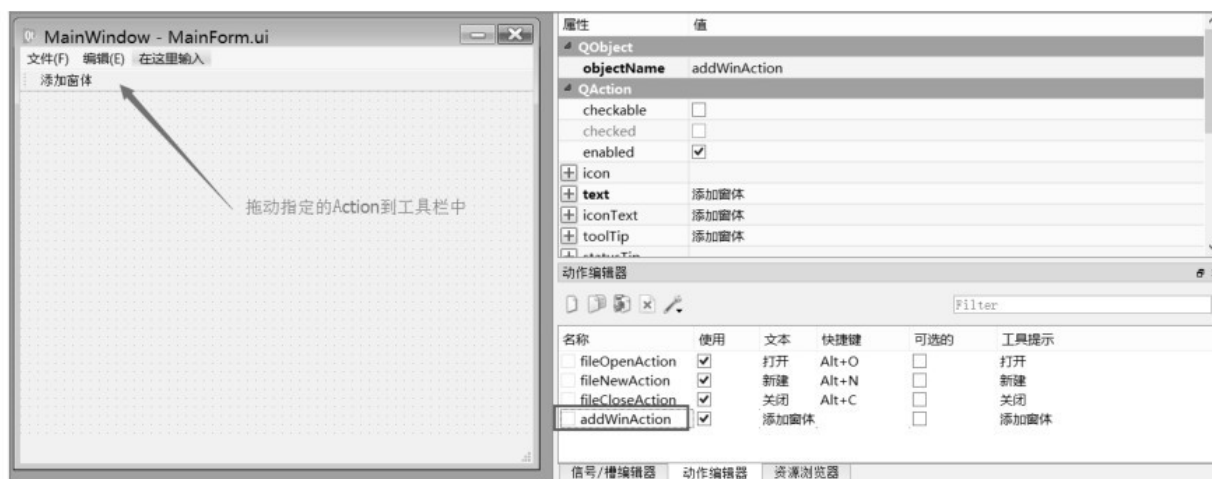


图3-64

在 Qt Designer 中，可以通过“动作编辑器”来配置动作的属性。

图3-1

对象名称	文 本	快 捷 键
fileOpenAction	打开	Alt + O
fileNewAction	新建	Alt + N
fileCloseAction	关闭	Alt + C
addWinAction	添加窗体	

pyuic5 UI Python

pyuic5-o MainForm.py MainForm.ui

MainForm.py PyQt5/Chapter03/mainWin

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(588, 476)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 588, 26))
        self.menubar.setObjectName("menubar")
        self.menu = QtWidgets.QMenu(self.menubar)
        self.menu.setObjectName("menu")
        self.menu_E = QtWidgets.QMenu(self.menubar)
        self.menu_E.setObjectName("menu_E")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)
        self.toolBar = QtWidgets.QToolBar(MainWindow)
        self.toolBar.setObjectName("toolBar")
        MainWindow.addToolBar(QtCore.Qt.TopToolBarArea, self.toolBar)
        self.fileOpenAction = QtWidgets.QAction(MainWindow)
        self.fileOpenAction.setObjectName("fileOpenAction")
        self.fileNewAction = QtWidgets.QAction(MainWindow)
        self.fileNewAction.setObjectName("fileNewAction")
        self.fileCloseAction = QtWidgets.QAction(MainWindow)
```



```

self.fileCloseAction.setObjectName("fileCloseAction")
self.addWinAction = QtWidgets.QAction(MainWindow)
self.addWinAction.setObjectName("addWinAction")
self.menu.addAction(self.fileOpenAction)
self.menu.addAction(self.fileNewAction)
self.menu.addAction(self.fileCloseAction)
self.menubar.addAction(self.menu.menuAction())
self.menubar.addAction(self.menu_E.menuAction())
self.toolBar.addAction(self.addWinAction)

self.retranslateUi(MainWindow)
QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):
    _translate = QtCore.QCoreApplication.translate
    MainWindow.setWindowTitle(_translate("MainWindow",
"MainWindow"))
    self.menu.setTitle(_translate("MainWindow", "文件(&F)"))
    self.menu_E.setTitle(_translate("MainWindow", "编辑(&E)"))
    self.toolBar.setWindowTitle(_translate("MainWindow",
"toolBar"))
    self.fileOpenAction.setText(_translate("MainWindow", "打开"))
    self.fileOpenAction.setShortcut(_translate("MainWindow",
"Alt+O"))
    self.fileNewAction.setText(_translate("MainWindow", "新建"))
    self.fileNewAction.setShortcut(_translate("MainWindow",
"Alt+N"))
    self.fileCloseAction.setText(_translate("MainWindow", "关闭"))
    self.fileCloseAction.setShortcut(_translate("MainWindow",
"Alt+C"))
    self.addWinAction.setText(_translate("MainWindow", "添加窗体"))

```

## 3.5.2 打包

打包后的文件结构如下所示，其中pyuic5是用于生成UI文件的工具，MainForm.ui是UI文件，MainForm.py是主程序文件，CallMainWin01.py是调用主程序的脚本文件。

## CallMainWin01.py PyQt5/Chapter03/mainWin

```
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget,
QFileDialog

from MainForm import Ui_MainWindow

class MainForm( QMainWindow , Ui_MainWindow):
    def __init__(self):
        super(MainForm,self).__init__()
        self.setupUi(self)
        # 菜单的点击事件, 当点击关闭菜单时连接槽函数 close()
        self.fileCloseAction.triggered.connect(self.close)
        # 菜单的点击事件, 当点击打开菜单时连接槽函数 openMsg()
        self.fileOpenAction.triggered.connect(self.openMsg)

    def openMsg(self):
        file,ok= QFileDialog.getOpenFileName(self,"打开","C:/","All
Files (*.*);;Text Files (*.txt)")
        # 在状态栏显示文件地址
        self.statusbar.showMessage(file)

if __name__=="__main__":
    app = QApplication(sys.argv)
    win = MainForm()
    win.show()
    sys.exit(app.exec_())
```

3-65

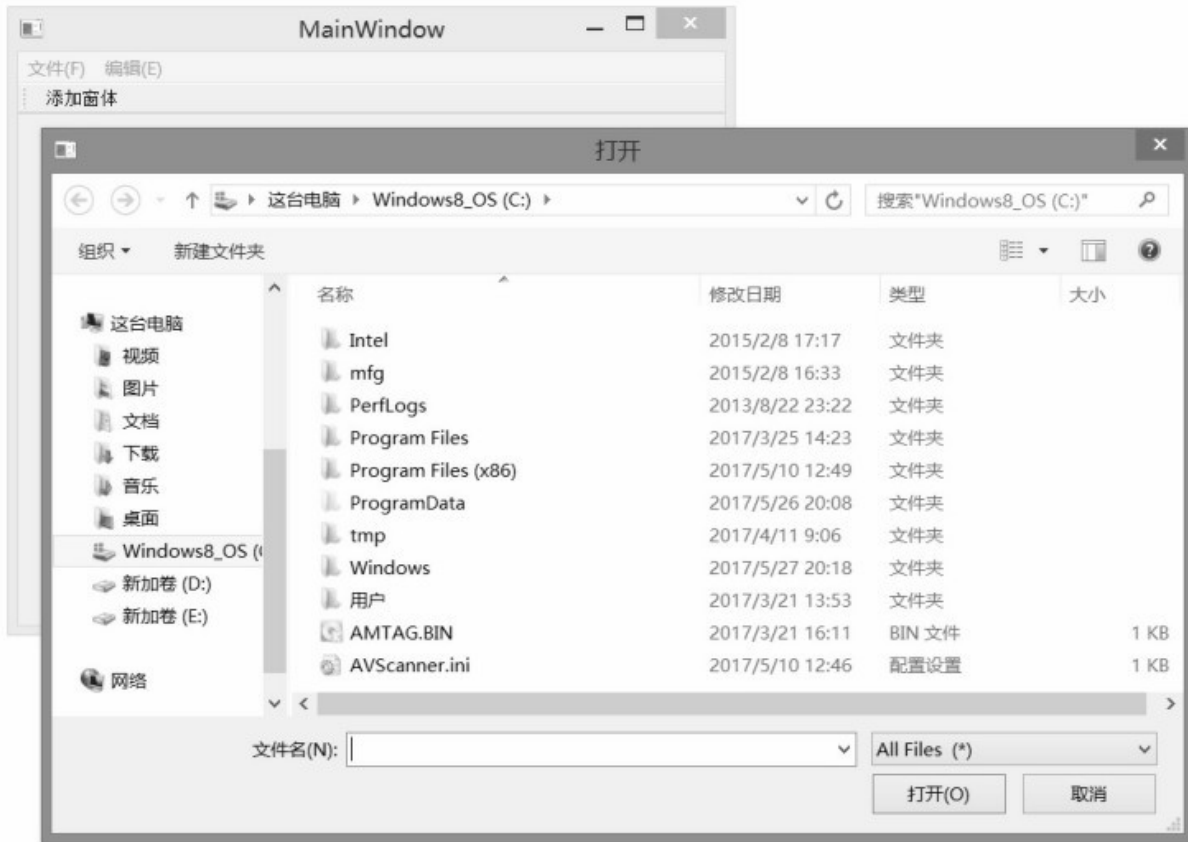


图3-65

在Qt Designer中，通过“文件”菜单，可以打开和保存文件。

```
# 打开文件
self.fileCloseAction.triggered.connect(self.close)

# 保存文件
self.fileOpenAction.triggered.connect(self.openMsg)
```

### 3.5.3 文本编辑

在Qt Designer中，可以通过“文件”菜单，打开和保存文件。在Qt Designer中，可以通过“文件”菜单，打开和保存文件。在Qt Designer中，可以通过“文件”菜单，打开和保存文件。

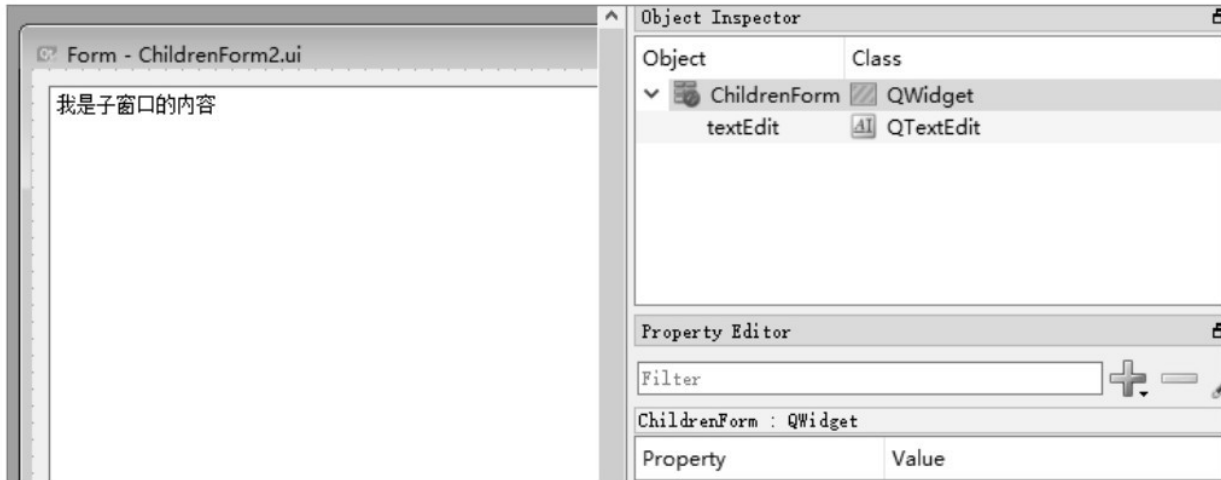


图3-66

在Qt Designer中，将MainForm.ui和MainForm2.ui都添加到“MaingridLayout”中，如图3-67所示。

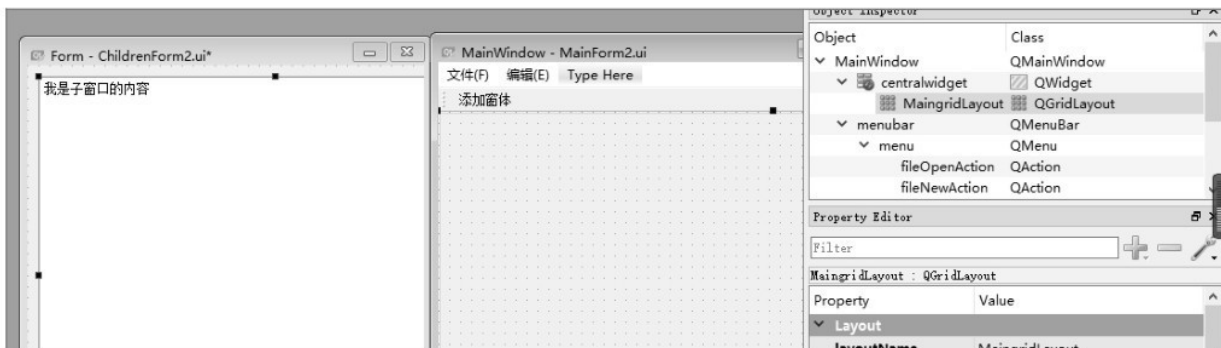


图3-67

在pyuic5.py文件中，添加以下代码：

```

pyuic5-o MainForm2.py MainForm2.ui
pyuic5-o ChildrenForm2.py ChildrenForm2.ui
MainForm2.py PyQt5/Chapter03/mainWin
3
CallMainWin02.py ChildrenForm2

```

CallMainWin02.py CallMainWin02.py  
PyQt5/Chapter03/mainWin

```

# -*- coding: utf-8 -*-

import sys
from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget,
QFileDialog
from MainForm2 import Ui_MainWindow
from ChildrenForm2 import Ui_ChildrenForm

class MainForm(QMainWindow, Ui_MainWindow):
    def __init__(self):
        super(MainForm, self).__init__()
        self.setupUi(self)

        # self.child = children() 生成子窗口实例 self.child
        self.child = ChildrenForm()

        # 菜单的单击事件, 当单击关闭菜单时连接槽函数 close()
        self.fileCloseAction.triggered.connect(self.close)
        # 菜单的单击事件, 当单击打开菜单时连接槽函数 openMsg()
        self.fileOpenAction.triggered.connect(self.openMsg)

        # 单击 actionTst, 子窗口就会显示在主窗口的 MaingridLayout 中
        self.addWinAction.triggered.connect(self.childShow)

    def childShow(self):
        # 添加子窗口
        self.MaingridLayout.addWidget(self.child)
        self.child.show()

    def openMsg(self):
        file, ok = QFileDialog.getOpenFileName(self, "打开", "C:/", "All
Files (*);;Text Files (*.txt)")
        # 在状态栏显示文件地址
        self.statusbar.showMessage(file)

class ChildrenForm(QWidget, Ui_ChildrenForm):
    def __init__(self):
        super(ChildrenForm, self).__init__()
        self.setupUi(self)

```

```
if __name__ == "__main__":  
    app = QApplication(sys.argv)  
    win = MainForm()  
    win.show()  
    sys.exit(app.exec_())
```

□□□□□□□□□□3-68□□□



□3-68

```

        mainWindow.setTitle("MainWindow")
        MainForm.addChildForm2()
        MainForm.childShow()
        # actionTstMainGridLayout
    
```



```

self.addAction.triggered.connect(self.childShow)
def childShow(self):
    # 子窗口
    self.MainGridLayout.addWidget(self.child)
    self.child.show()

```

## 3.6 资源管理

PyQt 5 资源管理是 Qt 资源管理的一部分。Python 资源管理 Python 资源管理 Qt Designer 资源管理 4-7 资源管理

### 3.6.1 Qt Designer 资源管理

Qt Designer 资源管理 PyQt 资源管理 .qrc 资源管理 .qrc 资源管理 1 资源管理 apprcc.qrc 资源管理 rcc version="1.0" qresource /qresource /rcc Eric 资源管理 3- 69 资源管理 apprcc.qrc 资源管理



图3-69

第2步 在 Qt Designer 中新建一个 Widget 窗口，命名为 MainWin02.ui，如图 3-70 所示。将资源文件复制到 PyQt5/Chapter03/images 文件夹。

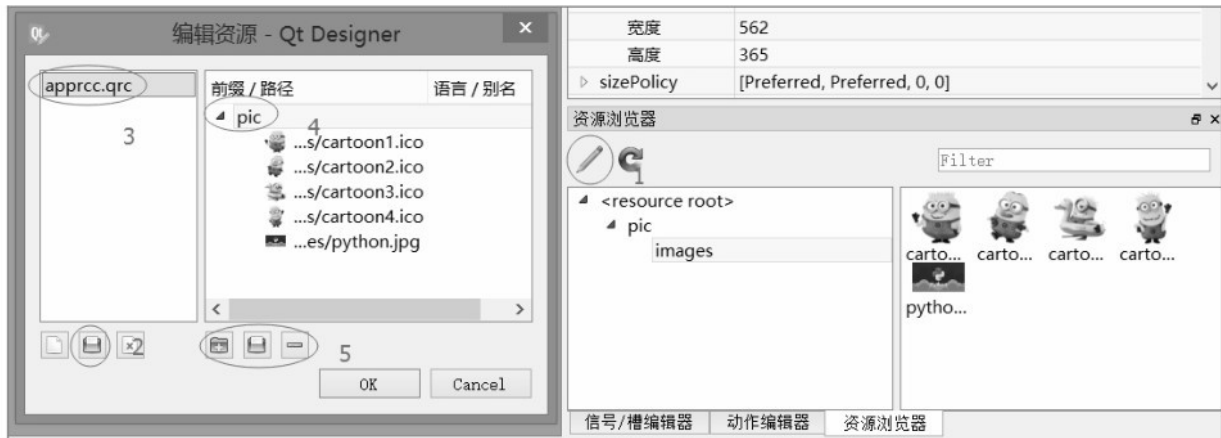


图3-70

在 Qt Designer 中，我们可以在“资源浏览器”中，通过“资源浏览器”来添加资源文件。在“资源浏览器”中，我们可以在“资源浏览器”中，通过“资源浏览器”来添加资源文件。在“资源浏览器”中，我们可以在“资源浏览器”中，通过“资源浏览器”来添加资源文件。

```

[qresource prefix="pic"
  [file images/cartoon1.ico]/file
  [file images/cartoon2.ico]/file
  [file images/cartoon3.ico]/file
  [file images/cartoon4.ico]/file
  [file images/python.jpg]/file
]/qresource
]/RCC

```

### 3.6.2 资源文件

#### 1. Qt Designer 资源文件

在 Qt Designer 中，我们可以在“资源浏览器”中，通过“资源浏览器”来添加资源文件。在“资源浏览器”中，我们可以在“资源浏览器”中，通过“资源浏览器”来添加资源文件。在“资源浏览器”中，我们可以在“资源浏览器”中，通过“资源浏览器”来添加资源文件。

Designer 添加 pixmap 属性

图 3-71

## 2. 编写 .ui 文件

使用 pyuic5 生成 .ui 文件

pyuic5-o MainWin02.py MainWin02.ui



图 3-71

在 PyQt5/Chapter03/MainWin02.py 中添加以下代码

```
from PyQt5 import QtCore,QtGui,QtWidgets
class Ui_Form(object):
    def setupUi(self,Form):
        Form.setObjectName("Form")
        Form.resize(678,431)
        self.label=QtWidgets.QLabel(Form)
        self.label.setGeometry(QtCore.QRect(80,30,531,321))
        self.label.setText("")
```

```

        self.label.setPixmap(QtGui.QPixmap(":/pic/images/
python.jpg"))
        self.label.setObjectName("label")
        self.retranslateUi(Form)
        QtCore.QMetaObject.connectSlotsByName(Form)
    def retranslateUi(self,Form):
        _translate=QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form","Form"))
import apprcc_rc
##### CallMain Win02.py#####
#####
import sys
from PyQt5.QtWidgets import QApplication
,QMainWindow
from MainWin02 import Ui_Form
class MyMainWindow(QMainWindow,Ui_Form):
    def __init__(self,parent=None):
        super(MyMainWindow,self).__init__(parent)
        self.setupUi(self)
if __name__=="__main__":
    app=QApplication(sys.argv)
    myWin=MyMainWindow()
    myWin.show()
    sys.exit(app.exec_())
##### CallMainWin02.py ##### MainWin02.py #####
#####apprcc_rc#####
Exception "unhandled ImportError"

```

No module named 'apprrcc\_rc'

#####

import apprrcc\_rc

#####.qrc#####.py#####.py#####

### 3.6.3 #####

PyQt 5#####pyrcc5 #####apprrcc.qrc#####apprrcc\_rc.py#####  
#####\_rc##### Qt Designer #####\_rc #####Qt  
Designer#####

pyrcc5 apprrcc.qrc-o apprrcc\_rc.py

#####.qrc #####.py #####apprrcc\_rc.py  
#####

```
from PyQt5 import QtCore
```

```
qt_resource_data = b"\  
\x00\x00\x42\x3e\  
\x00\x01\x00\x01\x00\x40\x40\x00\x00\x01\x00\x20\x00\x28\x42\x00\  

```







```

self.label.setGeometry(QRect(80,30,531,32
1))
self.label.setText("")
self.label.setPixmap(QtGui.QPixmap(":/pic/images/
python.jpg"))
self.label.setObjectName("label")
self.retranslateUi(Form)
QtCore.QMetaObject.connectSlotsByName(Form)
def retranslateUi(self,Form):
    _translate=QtCore.QCoreApplication.translate
    Form.setWindowTitle(_translate("Form","Form"))
import apprcc_rc
CallMainWin02.py3-73

```



□3-73

□□□□□□□□□□□□□□□□□□□□□□

## 4 PyQt 5

PyQt 5は、PythonでC++のQtフレームワークを簡単に利用できるように設計されたライブラリです。

—— PyQt 5のインストール

PyQt 5のインストールには、pipを使用して以下のコマンドを実行します。

### 4.1 QMainWindow

QMainWindowは、Qtのメインウィンドウクラスです。通常、アプリケーションのメインウィンドウとして使用されます。

#### 4.1.1 例

QMainWindow、QWidget、QDialogは、QtのGUIクラスです。

QMainWindowは、Qtのメインウィンドウクラスです。通常、アプリケーションのメインウィンドウとして使用されます。

QDialogは、Qtのダイアログボックスクラスです。通常、ユーザーからの入力を受け取るために使用されます。

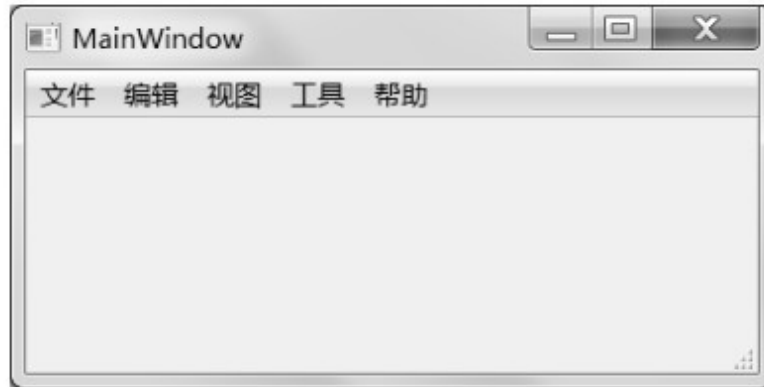


图4-1

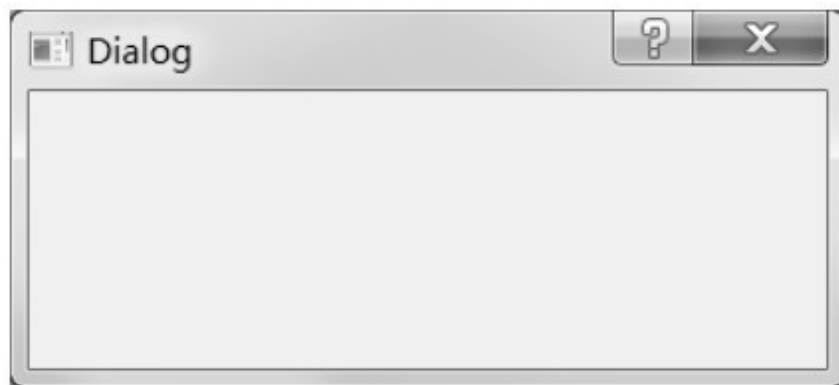


图4-2

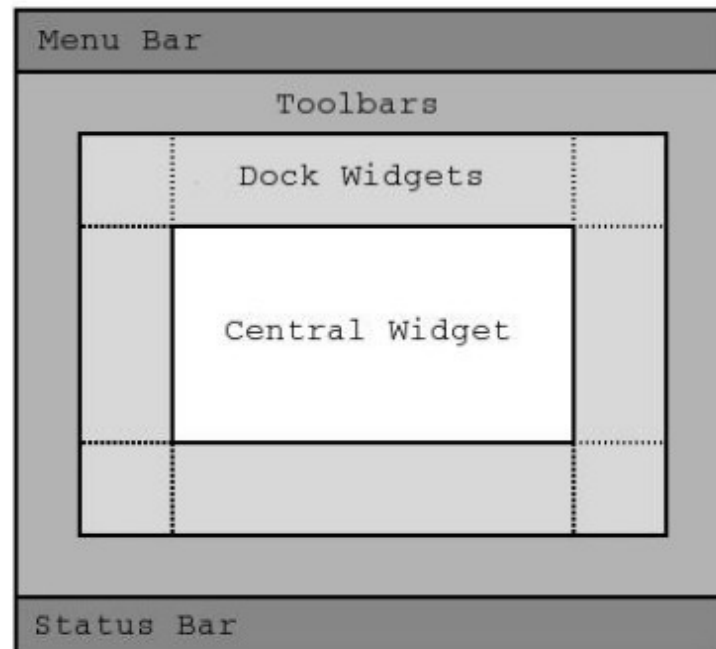
下面将分别介绍 QMainWindow 和 QDialog 类，以及 QWidget 类。

下面将分别介绍 QMainWindow 和 QDialog 类，以及 QWidget 类。

### 4.1.2 窗口类

下面将分别介绍 QMainWindow 和 QDialog 类，以及 QWidget 类。

PyQt QMainWindow QWidget  
setCentralWidget() 4-3



□4-3

```
QMainWindow □ □ QWidget □ □ □ □ □ □ □ □ □ □ □ □
```

## QWidget

1

## + - QMainWindow

## QMainWindow□□□□□□□□□□4-1□□□

□4-1

方 法	描 述
addToolBar()	添加工具栏
centralWidget()	返回窗口中心的一个控件，未设置时返回 NULL
menuBar()	返回主窗口的菜单栏
setCentralWidget()	设置窗口中心的控件
setStatusBar()	设置状态栏
statusBar()	获得状态栏对象后，调用状态栏对象的 showMessage(message, int timeout = 0)方法，显示状态栏信息。其中第一个参数是要显示的状态栏信息；第二个参数是信息停留的时间，单位是毫秒，默认是 0，表示一直显示状态栏信息

□□

QMainWindow□□□□□□□□setLayout()□□□□□□□□□□□□

### □□4-1 □□□□□

□□□□□□PyQt5/Chapter04/qt402\_QMainWin.py□□□□PyQt 5  
□□□□□□□□□□□□□□□□□□□□□□□□

```
import sys
from PyQt5.QtWidgets import QMainWindow
,QApplication
from PyQt5.QtGui import QIcon
class MainWindow(QMainWindow):
    def __init__(self,parent=None):
        super(MainWindow,self).__init__(parent)
        self.resize(400,200)
        self.status=self.statusBar()
        self.status.showMessage("□□□□□□□□",5000)
        self.setWindowTitle("PyQt MainWindow□□")
if __name__=="__main__":
    app=QApplication(sys.argv)
    app.setWindowIcon(QIcon("./images/cartoon1.ico"))
    form=MainWindow()
```

```
form.show()
sys.exit(app.exec_())
```

图4-4



图4-4

代码如下

```
self.status.showMessage("这是状态栏提示",5000)
```

在 QMainWindow 中添加 statusBar() 方法，调用 showMessage() 方法，可以在状态栏中显示提示信息。5 秒后自动消失。

MainWindow 类继承 QMainWindow 类，调用 QMainWindow 的 super() 方法，调用 statusBar() 方法，调用 showMessage() 方法。

### 4.1.3 状态栏提示

#### 图4-2 状态栏提示

MainWindow 类继承 QDesktopWidget 类，调用 QMainWindow 的 super() 方法，调用 statusBar() 方法，调用 showMessage() 方法。

□□□□□□

```
from PyQt5.QtWidgets import QDesktopWidget, QApplication ,QMainWindow
import sys

class Winform( QMainWindow):

    def __init__(self, parent=None):
        super( Winform, self).__init__(parent)

        self.setWindowTitle('主窗口放在屏幕中间例子')

        self.resize(370, 250)
        self.center()

    def center(self):
        screen = QDesktopWidget().screenGeometry()
        size = self.geometry()
        self.move((screen.width() - size.width()) / 2, (screen.height()
- size.height()) / 2)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    win = Winform()
    win.show()
    sys.exit(app.exec_())
```

□□□□□□□□□□4-5□□□



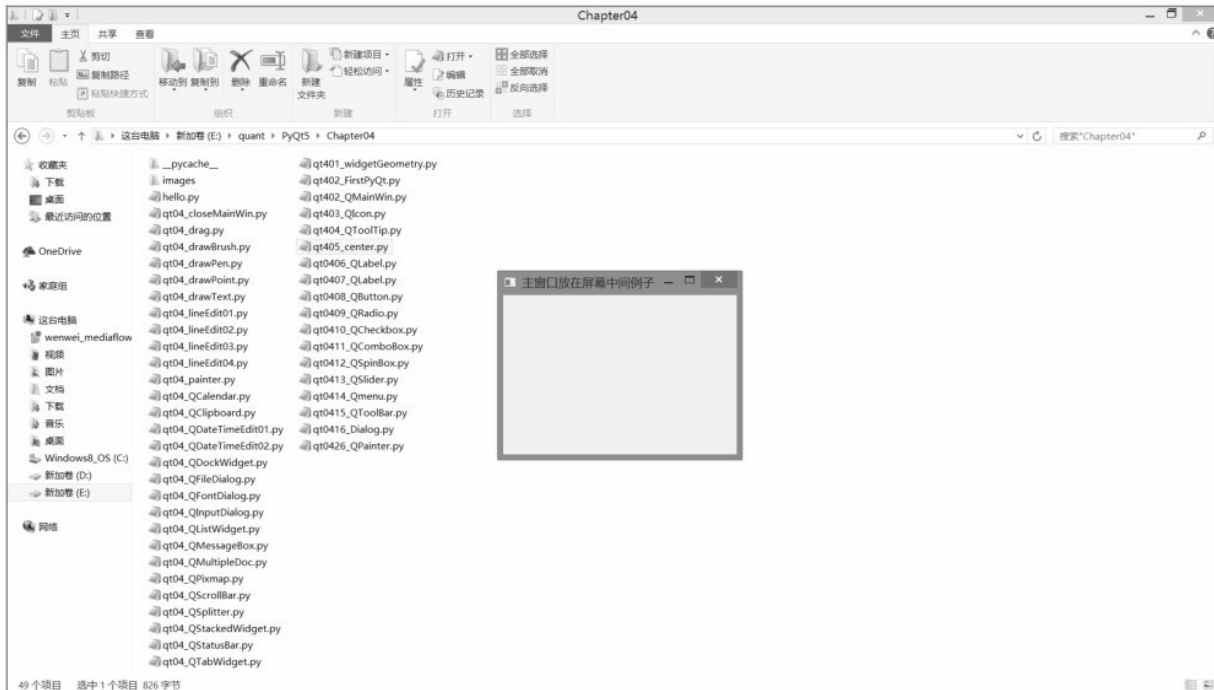


图4-5

```

self.resize(370,250)
QWidget.resize(370,250)
screen=QDesktopWidget().screenGeometry()
(screen.width()*screen.height())
QDesktopWidget
QDesktopWidget().screenGeometry()
size=self.geometry()
QWidget(size.width()*size.height())
self.move((screen.width()-size.width())/2,
(screen.height()-size.height())/2)

```

#### 4.1.4

### 例4-3 关闭窗口

例4-3 PyQt5/Chapter04/qt04\_closeMainWin.py 关闭窗口

```
from PyQt5.QtWidgets import QMainWindow, QHBoxLayout, QPushButton ,
QApplication, QWidget
import sys

class WinForm(QMainWindow):

    def __init__(self, parent=None):
        super(WinForm, self).__init__(parent)
        self.setWindowTitle('关闭窗口例子')
        self.button1 = QPushButton('关闭窗口')
        self.button1.clicked.connect(self.onButtonClick)

        layout = QHBoxLayout()
        layout.addWidget(self.button1)

        main_frame = QWidget()
        main_frame.setLayout(layout)
        self.setCentralWidget(main_frame)

    def onButtonClick(self ):
        # sender 是发送信号的对象
        sender = self.sender()
        print( sender.text() + ' 被按下了' )
        qApp = QApplication.instance()
        qApp.quit()
```

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = WinForm()
    form.show()
    sys.exit(app.exec_())
```

图4-6



图4-6

```

class MainWindow(QMainWindow):
    def __init__(self):
        super(MainWindow, self).__init__()
        self.button1.clicked.connect(self.onButtonClick)
        self.onButtonClick()
        QApplication.quit()
        self.setWindowTitle('关闭主窗口')
        self.button1.setText('关闭主窗口')

    def onButtonClick(self):
        # sender 返回的是发送信号的对象，即button1
        sender=self.sender()
        print( sender.text() + ' 被点击' )
        qApp=QApplication.instance()
        qApp.quit()

```

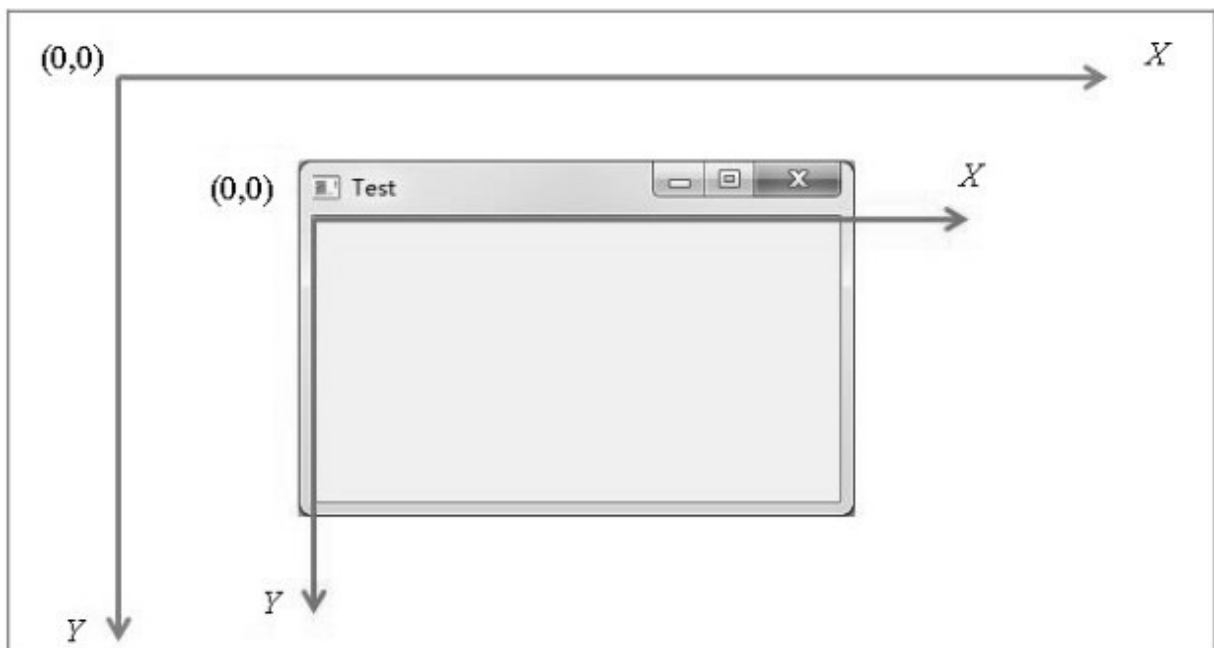
## 4.2 QWidget

QWidget를 상속받은 클래스를 QWidget으로  
QWidget

Widget은 “” PyQt PyQt  
PyQt PyQt  
PyQt PyQt  
PyQt PyQt  
PyQt PyQt  
PyQt PyQt

### 4.2.1

PyQt 4-7



4-7

(0,0) x y  
x y

x y Client Area Window Title Frame

圖 4-8 顯示 Qt 類別 `QWidget` 的幾何資訊“Window and Dialog Widgets”的幾何資訊。

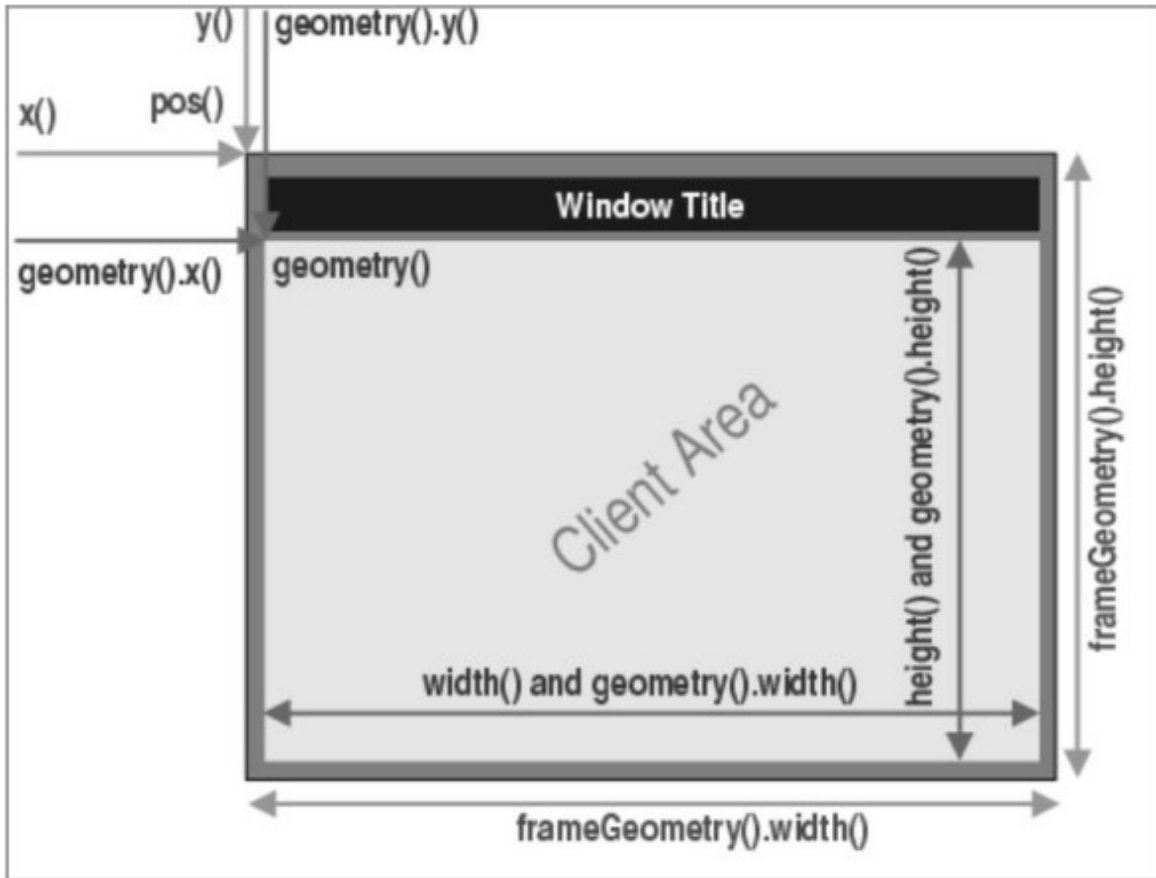


圖 4-8

圖 4-8 顯示 Qt 類別 `QWidget` 的幾何資訊。

`QWidget` 類別的 `x()`、`y()`、`width()` 和 `height()` 方法返回 widget 的幾何資訊。

`QWidget` 的 `geometry()` 方法返回 widget 的幾何資訊，包括 `width()` 和 `height()`。

`QWidget` 的 `frameGeometry()` 方法返回 widget 的幾何資訊，包括 `width()` 和 `height()`。

## 4.2.2 幾何資訊



`QWidget.setGeometry(int x,int y,int width,int height);`

`QWidget.setGeometry( QRect rect)`

`x y x y x y`

## 2. `QWidget`

`QWidget`

1

`QWidget.frameGeometry()`

2

`QWidget.move(int x,int y)`

`QWidget.move(QPoint point )`

3

`QWidget.pos()`

### [4-4](#)

PyQt5/Chapter04/qt401\_widgetGeometry.py

PyQt 5 `QWidget`

```
from PyQt5.QtWidgets import QApplication, QWidget, QPushButton
import sys

app = QApplication(sys.argv)
widget = QWidget()
btn = QPushButton( widget )
btn.setText("Button")
#以 QWidget 左上角为 (0, 0) 点
```

```

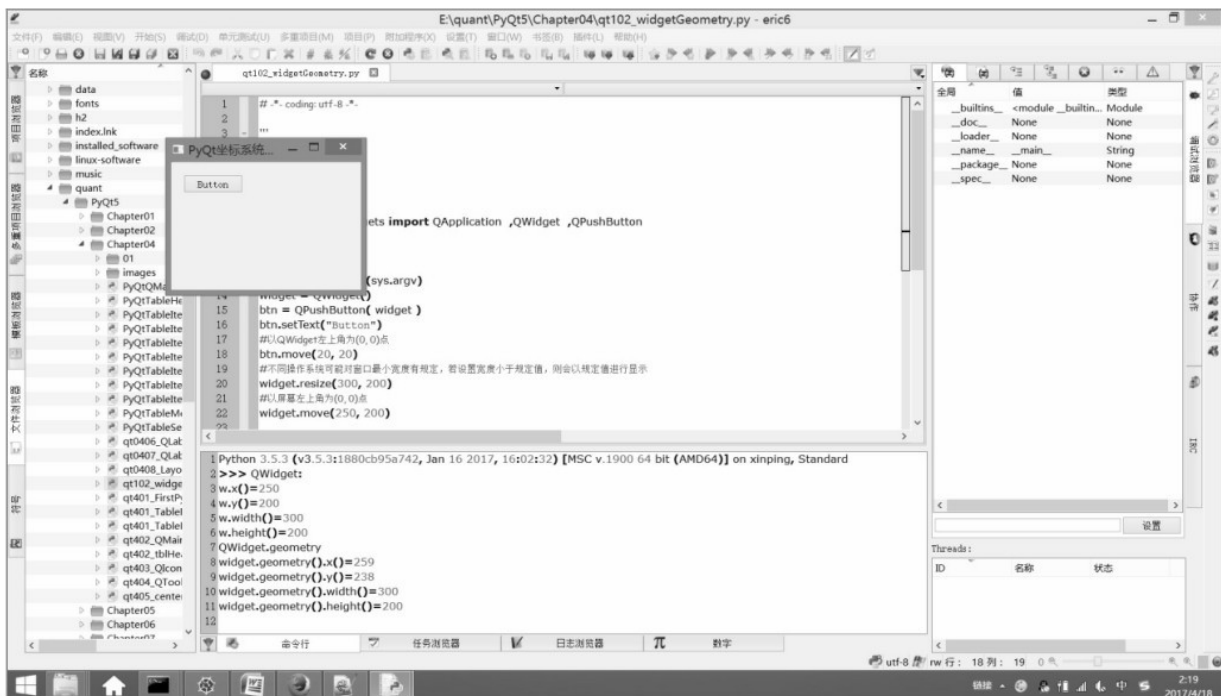
btn.move(20, 20)
#不同的操作系统可能对窗口的最小宽度有规定, 若设置宽度小于规定值, 则会以规定值进行显示
widget.resize(300, 200)
#以屏幕左上角为(0, 0)点
widget.move(250, 200)

widget.setWindowTitle('PyQt 坐标系统例子')
widget.show()
print("QWidget:")
print("w.x()=%d" % widget.x() )
print("w.y()=%d" % widget.y() )
print("w.width()=%d" % widget.width() )
print("w.height()=%d" % widget.height() )
print("QWidget.geometry")
print("widget.geometry().x()=%d" % widget.geometry().x() )
print("widget.geometry().y()=%d" % widget.geometry().y() )
print("widget.geometry().width()=%d" % widget.geometry().width() )
print("widget.geometry().height()=%d" % widget.geometry().height() )

sys.exit(app.exec_())

```

4-9





### 4.2.3 PyQt 5

PyQt5  
PyQt  
PyQt Win32  
MFC

4-5                        

PyQt5/Chapter04/qt402\_FirstPyQt.py PyQt 5

```
#-*- coding: UTF-8-*  
import sys  
from PyQt5.QtWidgets import QApplication, QWidget  
app=QApplication(sys.argv)  
window=QWidget()  
window.resize(300,200)  
window.move(250,150)  
window.setWindowTitle('Hello PyQt5')  
window.show()  
sys.exit(app.exec ())
```

□□□□□□□□□□4-10□□□

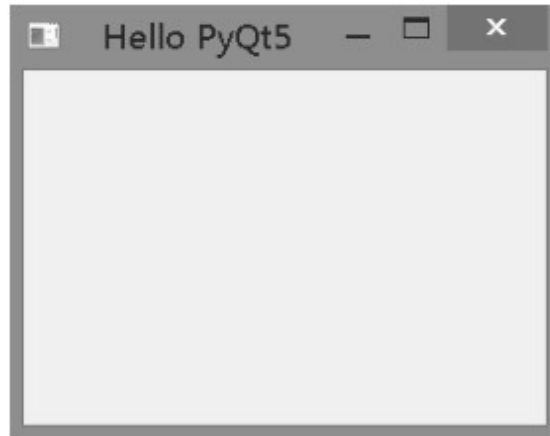


图4-10

代码如下

```
#-*- coding: UTF-8-*
```

PyQt PyQt

UTF-8?

UTF-8 8-bit Unicode Transformation Format Ken Thompson 1992 RFC 3629 UTF-8 1 4 Unicode PyQt Windows Linux

```
import sys
```

```
from PyQt5.QtWidgets import QApplication, QWidget
```

Qt5 GUI PyQt5.QtWidgets

```
app=QApplication(sys.argv)
```

PyQt5 QApplication QApplication sys.argv Python Shell .py

```
window=QWidget()
```

QWidget 是 PyQt5 中所有窗口类的基础类。它提供了窗口的基本功能，如设置窗口标题、图标、大小和位置等。

在 PyQt5 中，QWidget 类提供了以下方法：  
setWindowTitle() 设置窗口标题  
setWindowIcon() 设置窗口图标

resize(300,200)

resize() 方法用于设置窗口的宽度和高度。例如，将窗口大小设置为 300x200。

move(250,150)

move() 方法用于设置窗口的坐标 (x,y)。坐标的原点 (0,0) 位于窗口的左上角。此方法在 iOS、Android 和 Windows Phone 上均适用。

在 PyQt5 中，窗口的坐标 (0,0) 位于窗口的左上角。

x 坐标

y 坐标

4-11

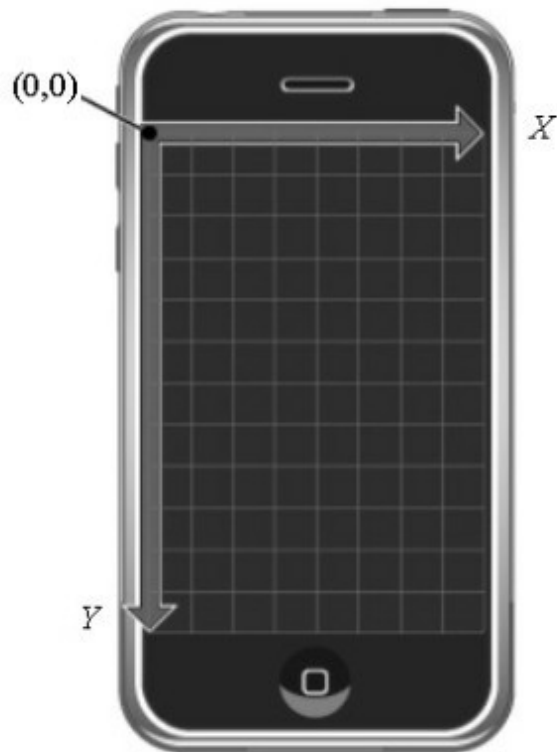


图4-11

```

window.setWindowTitle('Hello PyQt5')
# 设置窗口标题
window.show()
# 显示窗口
sys.exit(app.exec_())

# 退出程序
exit()  # 与 sys.exit() 类似
# 退出程序

# 使用 exec_() 方法
# 退出程序

# 使用 C++ 的 main 函数
# 退出程序
exit()
# 退出程序

```

- 返回0 EXIT\_SUCCESS 正常退出
- 返回EXIT\_FAILURE 异常退出
- 返回非0值 自定义退出

return 0 正常退出 return 1 异常退出 PyQt 5 使用  
C++ 的 exec\_() 返回0

exec\_() 返回非0

QApplication 使用 exec\_() 返回 PyQt 4 使用 Python 2 使用  
exec Python 使用 PyQt 5 使用 exec\_() 返回 Python 3  
Python 3 使用 Python 3 使用

app.exec()

返回非0 异常退出

#### 4.2.4 图标

PyQt 5 使用 PyQt 5 使用  
图标

PyQt 使用  
easyicon <http://www.easyicon.net/> easyicon  
50 PNG ICO ICNS 图标

#### 4-6 图标

PyQt5/Chapter04/qt403\_QIcon.py PyQt 5 使用  
图标

```

import sys
from PyQt5.QtGui import QIcon
from PyQt5.QtWidgets import QWidget , QApplication

#1 创建一个名为 Icon 的窗口类，继承自 QWidget 类
class Icon(QWidget):
    def __init__(self, parent = None):
        super(Icon,self).__init__(parent)
        self.initUI()

    #2 初始化窗口
    def initUI(self):
        self.setGeometry(300, 300, 250, 150)
        self.setWindowTitle('程序图标')
        self.setWindowIcon(QIcon('./images/cartoon1.ico'))

if __name__ == '__main__':
    app = QApplication(sys.argv)

```

```

icon = Icon()
icon.show()
sys.exit(app.exec_())

```

## 图4-12



图4-12

PyQt 5 是一个跨平台的 Python 图形用户界面库，它基于 Qt 框架。Python 是一个解释型、面向对象、动态数据类型的高级语言。PyQt 是一个 Python bindings for Qt 的库，它允许 Python 程序使用 Qt 的 GUI 功能。

```

1 QIcon(QIcon::fromTheme(QStringLiteral("qtdesignericon")))
2 QWidget *w = new QWidget;
3 w->setWindowIcon(QIcon::fromTheme(QStringLiteral("qtdesignericon")));
4

```

```
self.setWindowTitle('PyQt5.QtGui.QIcon')  
self.setIcon(QIcon())  
from PyQt5.QtGui import QIcon
```

PyQt5/Chapter04/qt404\_QToolTip.py PyQt 5

```
import sys
from PyQt5.QtWidgets import QWidget, QToolTip , QApplication
from PyQt5.QtGui import QFont

class Winform(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()
```

```

def initUI(self):
    QToolTip.setFont(QFont('SansSerif', 10))
    self.setToolTip('这是一个<b>气泡提示</b>')
    self.setGeometry(200, 300, 400, 400)
    self.setWindowTitle('气泡提示 demo')

if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Winform ()
    win.show()
    sys.exit(app.exec_())

```

图4-13



图4-13

图4-13

图4-13



```

self.setToolTip('b/b')
.setToolTip()
QToolTip.setFont(QFont('SansSerif',10))

```

## 4.3 QLabel

QLabel은 GIF 이미지를 표시할 수 있는 클래스입니다. QLabel은 QWidget의 하위 클래스이며, QLabel은 QLabel의 하위 클래스입니다.



QLabel은 4-2

方 法	描 述
setAlignment()	按固定值方式对齐文本： <ul style="list-style-type: none"><li>• Qt.AlignLeft, 水平方向靠左对齐</li><li>• Qt.AlignRight, 水平方向靠右对齐</li><li>• Qt.AlignCenter, 水平方向居中对齐</li><li>• Qt.AlignJustify, 水平方向调整间距两端对齐</li><li>• Qt.AlignTop, 垂直方向靠上对齐</li><li>• Qt.AlignBottom, 垂直方向靠下对齐</li><li>• Qt.AlignVCenter, 垂直方向居中对齐</li></ul>
setIndent()	设置文本缩进值
setPixmap()	设置 QLabel 为一个 Pixmap 图片
text()	获得 QLabel 的文本内容
setText()	设置 QLabel 的文本内容
selectedText()	返回所选择的字符
setBuddy()	设置 QLabel 的助记符及 buddy（伙伴），即使用 QLabel 设置快捷键，会在快捷键后将焦点设置到其 buddy 上，这里用到了 QLabel 的交互控件功能。此外，buddy 可以是任何一个 Widget 控件。使用 setBuddy(QWidget *)设置，其 QLabel 必须是文本内容，并且使用“&”符号设置了助记符
setWordWrap()	设置是否允许换行

QLabel

4-3

信 号	描 述
linkActivated	当单击标签中嵌入的超链接，希望在新窗口中打开这个超链接时，setOpenExternalLinks 特性必须设置为 true
linkHovered	当鼠标指针滑过标签中嵌入的超链接时，需要用槽函数与这个信号进行绑定

4-7 QLabel

PyQt5/Chapter04/qt0406\_QLabel.py PyQt 5  
QLabel

```

from PyQt5.QtWidgets import QApplication, QLabel ,QWidget, QVBoxLayout
from PyQt5.QtCore import Qt
from PyQt5.QtGui import QPixmap ,QPalette
import sys

class WindowDemo(QWidget):
    def __init__(self ):
        super().__init__()

        label1 = QLabel(self)
        label2 = QLabel(self)
        label3 = QLabel(self)
        label4 = QLabel(self)

        #1 初始化标签控件
        label1.setText("这是一个文本标签。")
        label1.setAutoFillBackground(True)
        palette = QPalette()
        palette.setColor(QPalette.Window,Qt.blue)
        label1.setPalette(palette)
        label1.setAlignment( Qt.AlignCenter)

        label2.setText("<a href='#'>欢迎使用 Python GUI 应用</a>")

        label3.setAlignment( Qt.AlignCenter)
        label3.setToolTip('这是一个图片标签')
        label3.setPixmap( QPixmap("./images/python.jpg"))

        label4.setText("<A href='http://www.cnblogs.com/wangshuo1/'>欢
欢迎访问信平的小屋</a>")
        label4.setAlignment( Qt.AlignRight)
        label4.setToolTip('这是一个超链接标签')

        #2 在窗口布局中添加控件
        vbox=QVBoxLayout()
        vbox.addWidget(label1)
        vbox.addStretch()
        vbox.addWidget(label2)
        vbox.addStretch()

```

```

vbox.addWidget( label3 )
vbox.addStretch()
vbox.addWidget( label4)

#3 允许 label1 控件访问超链接
label1.setOpenExternalLinks(True)
# 打开允许访问超链接,默认是不允许,需要使用 setOpenExternalLinks(True)
允许浏览器访问超链接

label4.setOpenExternalLinks( False )
# 点击文本框绑定槽事件
label4.linkActivated.connect( link_clicked )

# 滑过文本框绑定槽事件
label2.linkHovered.connect( link_hovered )
label1.setTextInteractionFlags( Qt.TextSelectableByMouse )

self.setLayout(vbox)
self.setWindowTitle("QLabel 例子")

def link_hovered():
    print("当鼠标滑过 label-2 标签时, 触发事件。")

def link_clicked():
    print("当用鼠标点击 label-4 标签时, 触发事件。" )

if __name__ == "__main__":
    app = QApplication(sys.argv)
    win = WindowDemo()
    win.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□4-14□□□



图4-14

```

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        label1.setAlignment( Qt.AlignCenter)
        label2.setText("Python")
        label4.setText("欢迎访问信平的小屋")
        label4.setOpenExternalLinks(True)
        label4.linkHovered.connect(link_clicked)
        label4=QLabel(self)
        label4.setOpenExternalLinks( True )
        label4.setText("欢迎访问信平的小屋")
        href='http://www.cnblogs.com/wangshuo1/'

```

)

label4.linkActivated.connect( link\_clicked )

## 4-8 QLabel

PyQt5/Chapter04/qt0407\_QLabel.py PyQt 5  
QLabel

```
from PyQt5.QtWidgets import *
import sys

class QlabelDemo(QDialog):
    def __init__(self):
        super().__init__()

        self.setWindowTitle('QLabel 例子')
        nameLb1 = QLabel('&Name', self)
        nameEd1 = QLineEdit( self )
        nameLb1.setBuddy(nameEd1)

        nameLb2 = QLabel('&Password', self)
        nameEd2 = QLineEdit( self )
        nameLb2.setBuddy(nameEd2)

        btnOk = QPushButton('&OK')
        btnCancel = QPushButton('&Cancel')
        mainLayout = QGridLayout(self)
        mainLayout.addWidget(nameLb1,0,0)
```

```

        mainLayout.addWidget(nameEd1,0,1,1,2)

        mainLayout.addWidget(nameLb2,1,0)
        mainLayout.addWidget(nameEd2,1,1,1,2)

        mainLayout.addWidget(btnOk,2,1)
        mainLayout.addWidget(btnCancel,2,2)

def link_hovered():
    print("当鼠标滑过 label-2 标签时，触发事件。")

def link_clicked():
    print("当用鼠标点击 label-4 标签时，触发事件。" )

if __name__ == "__main__":
    app = QApplication(sys.argv)
    labelDemo = QlabelDemo()
    labelDemo.show()
    sys.exit(app.exec_())

```

图4-15



图4-15

图4-15展示了“Alt+N”快捷键的设置。在Qt中，可以通过调用QLabel的setShortcut方法来实现。以下代码展示了如何设置快捷键：

```

nameLb1=QLabel('&Name',self)
nameEd1=QLineEdit( self )
nameLb1.setBuddy(nameEd1)

```

## 4.4

### 4.4.1 QLineEdit

QLineEdit는 단일 줄 텍스트를 입력할 수 있는 widget이다. QTextEdit와 달리 QLineEdit은 한 줄만 입력할 수 있다. QLineEdit은 4-4와 같다.

4-4



方 法	描 述
setAlignment()	按固定值方式对齐文本： <ul style="list-style-type: none"> <li>• Qt.AlignLeft，水平方向靠左对齐</li> <li>• Qt.AlignRight，水平方向靠右对齐</li> <li>• Qt.AlignCenter，水平方向居中对齐</li> <li>• Qt.AlignJustify，水平方向调整间距两端对齐</li> <li>• Qt.AlignTop，垂直方向靠上对齐</li> <li>• Qt.AlignBottom，垂直方向靠下对齐</li> <li>• Qt.AlignVCenter，垂直方向居中对齐</li> </ul>
clear()	清除文本框内容
setEchoMode()	设置文本框显示格式。允许输入的文本显示格式的值可以是： <ul style="list-style-type: none"> <li>• QLineEdit.Normal，正常显示所输入的字符，此为默认选项</li> <li>• QLineEdit.NoEcho，不显示任何输入的字符，常用于密码类型的输入，且其密码长度需要保密时</li> <li>• QLineEdit.Password，显示与平台相关的密码掩码字符，而不是实际输入的字符</li> <li>• QLineEdit.PasswordEchoOnEdit，在编辑时显示字符，负责显示密码类型的输入</li> </ul>
setPlaceholderText()	设置文本框浮显文字
setMaxLength()	设置文本框所允许输入的最大字符数
setReadOnly()	设置文本框为只读的
setText()	设置文本框内容
Text()	返回文本框内容
setDragEnabled()	设置文本框是否接受拖动
setMaxLength()	设置允许输入字符的最大长度
selectAll()	全选
setFocus()	得到焦点
setInputMask()	设置掩码
setValidator()	设置文本框的验证器（验证规则），将限制任意可能输入的文本。可用的校验器为： <ul style="list-style-type: none"> <li>• QIntValidator，限制输入整数</li> <li>• QDoubleValidator，限制输入浮点数</li> <li>• QRegExpValidator，检查输入是否符合正则表达式</li> </ul>

□□□□□□□□□□ 4-5 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□



## [4-9 EchoMode](#)

PyQt5/Chapter04/qt04\_lineEdit01.py

```

from PyQt5.QtWidgets import QApplication, QLineEdit , QWidget ,
QFormLayout
import sys

class lineEditDemo(QWidget):
    def __init__(self, parent=None):
        super(lineEditDemo, self).__init__(parent)
        self.setWindowTitle("QLineEdit 例子")

        flo = QFormLayout()
        pNormalLineEdit = QLineEdit( )
        pNoEchoLineEdit = QLineEdit()
        pPasswordLineEdit = QLineEdit( )
        pPasswordEchoOnEditLineEdit = QLineEdit( )

        flo.addRow("Normal", pNormalLineEdit)
        flo.addRow("NoEcho", pNoEchoLineEdit)
        flo.addRow("Password", pPasswordLineEdit)
        flo.addRow("PasswordEchoOnEdit", pPasswordEchoOnEditLineEdit)

        pNormalLineEdit.setPlaceholderText("Normal")
        pNoEchoLineEdit.setPlaceholderText("NoEcho")
        pPasswordLineEdit.setPlaceholderText("Password")
        pPasswordEchoOnEditLineEdit.setPlaceholderText
("PasswordEchoOnEdit")

        # 设置显示效果
        pNormalLineEdit.setEchoMode(QLineEdit.Normal)
        pNoEchoLineEdit.setEchoMode(QLineEdit.NoEcho)
        pPasswordLineEdit.setEchoMode(QLineEdit.Password)
        pPasswordEchoOnEditLineEdit.setEchoMode
(QLineEdit.PasswordEchoOnEdit)

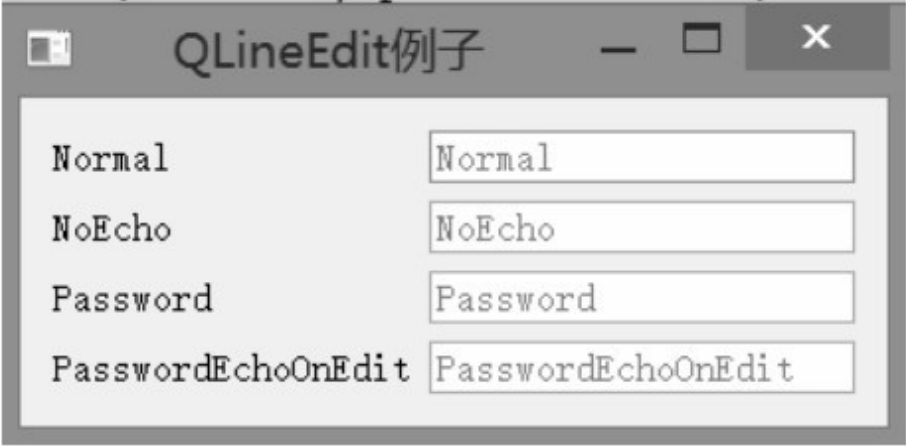
        self.setLayout(flo)

if __name__ == "__main__":
    app = QApplication(sys.argv)

```

```
win = lineEditDemo()
win.show()
sys.exit(app.exec_())
```

□□□□□□□□□□4-16□□.



□4-16

4-10

```
PyQt5/Chapter04/qt04_lineEdit02.py
```

```
from PyQt5.QtWidgets import QApplication, QLineEdit , QWidget ,
QFormLayout
from PyQt5.QtGui import QIntValidator ,QDoubleValidator ,
QRegExpValidator
from PyQt5.QtCore import QRegExp
import sys

class lineEditDemo(QWidget):
    def __init__(self, parent=None):
        super(lineEditDemo, self).__init__(parent)
        self.setWindowTitle("QLineEdit 例子")

        flo = QFormLayout()
        pIntLineEdit = QLineEdit()
        pDoubleLineEdit = QLineEdit()
        pValidatorLineEdit = QLineEdit()

        flo.addRow("整型", pIntLineEdit)
        flo.addRow("浮点型", pDoubleLineEdit)
        flo.addRow("字母和数字", pValidatorLineEdit)
```

```

pIntLineEdit.setPlaceholderText("整型")
pDoubleLineEdit.setPlaceholderText("浮点型")
pValidatorLineEdit.setPlaceholderText("字母和数字")

# 整型, 范围: [1, 99]
pIntValidator = QIntValidator(self)
pIntValidator.setRange(1, 99)

# 浮点型, 范围: [-360, 360], 精度: 小数点后两位
pDoubleValidator = QDoubleValidator(self)
pDoubleValidator.setRange(-360, 360)
pDoubleValidator.setNotation
(QDoubleValidator.StandardNotation)
pDoubleValidator.setDecimals(2)

# 字母和数字
reg = QRegExp("[a-zA-Z0-9]+$")
pValidator = QRegExpValidator(self)
pValidator.setRegExp(reg)

# 设置验证器
pIntLineEdit.setValidator(pIntValidator)
pDoubleLineEdit.setValidator(pDoubleValidator)
pValidatorLineEdit.setValidator(pValidator)

self.setLayout(flo)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    win = lineEditDemo()
    win.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□4-17□□□□

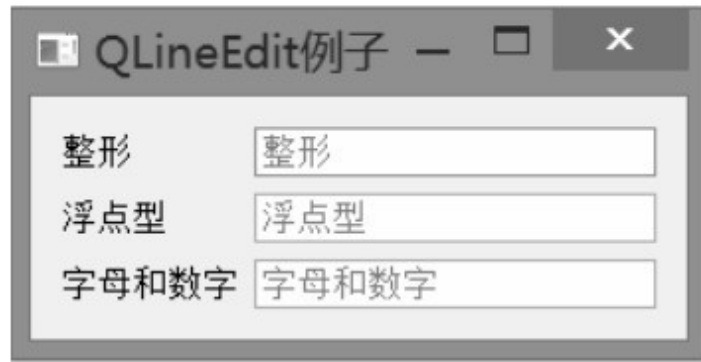


图4-17

### [图4-11](#)

验证IP地址和MAC地址的合法性。该程序位于PyQt5/Chapter04/qt04\_lineEdit03.py。



```

from PyQt5.QtWidgets import QApplication, QLineEdit , QWidget ,
QFormLayout
import sys

class lineEditDemo(QWidget):
    def __init__(self, parent=None):
        super(lineEditDemo, self).__init__(parent)
        self.setWindowTitle("QLineEdit 的输入掩码例子")

        flo = QFormLayout()
        pIPLineEdit = QLineEdit()
        pMACLineEdit = QLineEdit()
        pDateLineEdit = QLineEdit()
        pLicenseLineEdit = QLineEdit()

        pIPLineEdit.setInputMask("000.000.000.000;_")
        pMACLineEdit.setInputMask("HH:HH:HH:HH:HH:HH;_")
        pDateLineEdit.setInputMask("0000-00-00")
        pLicenseLineEdit.setInputMask(
">AAAAA-AAAAA-AAAAA-AAAAA-AAAAA;#")

        flo.addRow("数字掩码", pIPLineEdit)
        flo.addRow("Mac 掩码", pMACLineEdit)
        flo.addRow("日期掩码", pDateLineEdit)
        flo.addRow("许可证掩码", pLicenseLineEdit)

        self.setLayout(flo)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    win = lineEditDemo()
    win.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□4-18□□□

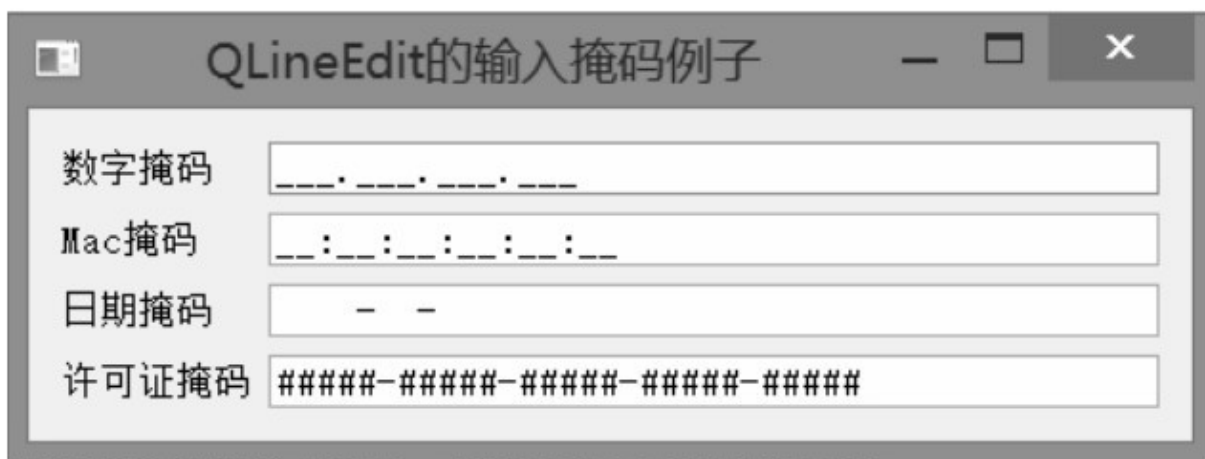


图4-18

## [图4-12 图例](#)

图例 PyQt5/Chapter04/qt04\_lineEdit04.py 图例  
QLabel 图例

```
from PyQt5.QtWidgets import QApplication, QLineEdit , QWidget ,
QFormLayout

from PyQt5.QtGui import QIntValidator , QDoubleValidator , QFont
from PyQt5.QtCore import Qt
import sys

class lineEditDemo(QWidget):
    def __init__(self, parent=None):
        super(lineEditDemo, self).__init__(parent)
        e1 = QLineEdit()
        e1.setValidator( QIntValidator() )
        e1.setMaxLength(4)
        e1.setAlignment( Qt.AlignRight )
        e1.setFont( QFont("Arial",20))
        e2 = QLineEdit()
        e2.setValidator( QDoubleValidator(0.99,99.99,2))
        flo = QFormLayout()
        flo.addRow("integer validator", e1)
        flo.addRow("Double validator",e2)
        e3 = QLineEdit()
        e3.setInputMask('+99_9999_999999')
        flo.addRow("Input Mask",e3)
        e4 = QLineEdit()
        e4.textChanged.connect( self.textchanged )
        flo.addRow("Text changed",e4)
        e5 = QLineEdit()
```

```

        e5.setEchoMode( QLineEdit.Password )
        flo.addRow("Password",e5)
        e6 = QLineEdit("Hello PyQt5")
        e6.setReadOnly(True)
        flo.addRow("Read Only",e6 )
        e5.editingFinished.connect( self.enterPress )
        self.setLayout(flo)
        self.setWindowTitle("QLineEdit 例子")

    def textchanged(self, text):
        print( "输入的内容为: "+text )

    def enterPress( self ):
        print( "已输入值" )

if __name__ == "__main__":
    app = QApplication(sys.argv)
    win = lineEditDemo()
    win.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□4-19□□□



```

class QLineEdit(QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setObjectName('lineEdit')
        self.setText('')
        self.setEchoMode(QLineEdit.EchoMode.Password)
        self.editingFinished.connect(self.enterPress)

    def enterPress(self):
        # 这里可以添加一些逻辑

```

#### 4.4.2 QTextEdit

QTextEdit 是多行文本框，它提供了丰富的文本编辑功能，如撤销、重做、复制、粘贴等。它支持 HTML 格式，可以显示和编辑富文本内容。QTextEdit 的类层次结构如图 4-8 所示。

图4-8

方 法	描 述
setPlainText()	设置多行文本框的文本内容。
toPlainText()	返回多行文本框的文本内容。
setHtml()	设置多行文本框的内容为 HTML 文档，HTML 文档是描述网页的。
toHtml()	返回多行文本框的 HTML 文档内容。
clear()	清除多行文本框的内容

#### 图4-13 QTextEdit

下面是一个使用 PyQt5/Chapter04/qt04\_textEdit.py 的例子，展示了 PyQt 5 中的 QTextEdit 类。



□□□□□□



□4-20



□4-21



图4-22

```

class QTextEditExample(QWidget):
    def __init__(self):
        super().__init__()
        self.textEdit = QTextEdit()
        self.btnPress1 = QPushButton("显示文本")
        self.btnPress2 = QPushButton("显示HTML")
        self.btnPress1.clicked.connect(self.btnPress1_Clicked)
        self.btnPress2.clicked.connect(self.btnPress2_Clicked)
        self.textEdit.setText("Hello PyQt5! 单击按钮。")
        self.layout = QVBoxLayout()
        self.layout.addWidget(self.textEdit)
        self.layout.addWidget(self.btnPress1)
        self.layout.addWidget(self.btnPress2)
        self.setLayout(self.layout)

    def btnPress1_Clicked(self):
        self.textEdit.setText(self.textEdit.toPlainText())

    def btnPress2_Clicked(self):
        self.textEdit.setText(self.textEdit.toHtml())

```

## 4.5 按钮

### 4.5.1 QAbstractButton

在Qt GUI中，按钮是最基本的控件之一。Qt提供了许多不同类型的按钮，如QPushButton、QCheckBox、QRadioButton等。这些按钮都继承自QAbstractButton类。在PyQt中，我们可以使用QAbstractButton类来创建自定义的按钮。





图4-11

方 法	描 述
setCheckable()	设置按钮是否已经被选中，如果设置为 True，则表示按钮将保持已点击和释放状态
toggle()	在按钮状态之间进行切换
setIcon()	设置按钮上的图标
setEnabled()	设置按钮是否可以使用，当设置为 False 时，按钮变成不可用状态，点击它不会发射信号
isChecked()	返回按钮的状态。返回值为 True 或 False
setDefault()	设置按钮的默认状态
setText()	设置按钮的显示文本
text()	返回按钮的显示文本

2. QPushButton

QPushButton 的文本为 “&Download”，即按下 Alt+D 时，将焦点移动到该按钮上。D 后面的 “&” 表示 D 为快捷键。D 后面的 “&&” 表示 D 为快捷键。QPushButton 的文本为 “&Download”，即按下 Alt+D 时，将焦点移动到该按钮上。D 后面的 “&” 表示 D 为快捷键。D 后面的 “&&” 表示 D 为快捷键。

```
self.button=QPushButton("&Download")
self.button.setDefault(True)
```

图4-23

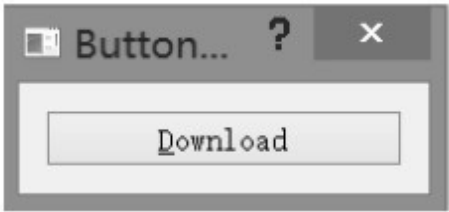


图4-23

4-14 QPushButton

PyQt5/Chapter04/qt0408\_QButton.py PyQt 5 QPushButton

```
import sys
from PyQt5.QtCore import *
```

```

from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class Form(QDialog):
    def __init__(self, parent=None):
        super(Form, self).__init__(parent)
        layout = QVBoxLayout()

        self.btn1 = QPushButton("Button1")
        self.btn1.setCheckable(True)
        self.btn1.toggle()
        self.btn1.clicked.connect(lambda:self.whichbtn(self.btn1) )
        self.btn1.clicked.connect(self.btnstate)
        layout.addWidget(self.btn1)

        self.btn2 = QPushButton('image')
        self.btn2.setIcon(QIcon(QPixmap("../images/python.png")))
        self.btn2.clicked.connect(lambda:self.whichbtn(self.btn2) )
        layout.addWidget(self.btn2)
        self.setLayout(layout)

        self.btn3 = QPushButton("Disabled")
        self.btn3.setEnabled(False)
        layout.addWidget(self.btn3)

        self.btn4= QPushButton("&Download")
        self.btn4.setDefault(True)
        self.btn4.clicked.connect(lambda:self.whichbtn(self.btn4))
        layout.addWidget(self.btn4)
        self.setWindowTitle("Button demo")

    def btnstate(self):
        if self.btn1.isChecked():
            print("button pressed" )
        else:
            print("button released" )

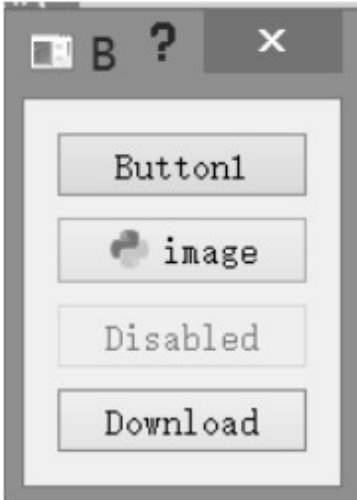
    def whichbtn(self,btn):
        print("clicked button is " + btn.text() )

if __name__ == '__main__':
    app = QApplication(sys.argv)

```

```
btnDemo = Form()
btnDemo.show()
sys.exit(app.exec ())
```

4-24



□4-24

□ □ □ □ □

```
□□□□□□□□ btn1 □ btn2 □ btn3 □ btn4 □ □ □ □ □ □ QPushButton
```

Clicked

1 btn1 toggle()

```
self.btn1=QPushButton("Button1")
```

```
self.btn1.setCheckable(True)
```

```
self.btn1.toggle()
```

```
clicked btnstate() btn.isChecked
```

□ □

```
self.btn1.clicked.connect(self.btnstate)
```

lambda btn1 clicked

`whichbtn()`



图4-12

方 法	描 述
setCheckable()	设置按钮是否已经被选中，可以改变单选钮的选中状态，如果设置为 True，则表示单选钮将保持已点击和释放状态
isChecked()	返回单选钮的状态。返回值为 True 或 False
setText()	设置单选钮的显示文本
text()	返回单选钮的显示文本

1 `QRadioButton` 的 `toggled` 信号在单选钮被选中或释放时发出。  
 2 该信号是一个布尔值，表示单选钮是否被选中。  
 3 `clicked` 信号在单选钮被点击时发出。  
 4 该信号是一个布尔值，表示单选钮是否被选中。  
 5 `toggled` 信号在单选钮被选中或释放时发出。

#### 图4-15 QRadioButton 示例

1 该示例位于 `PyQt5/Chapter04/qt0409_QRadio.py` 文件中。  
 2 该示例展示了 `PyQt 5` 中的 `QRadioButton` 控件。

```

import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class Radiodemo(QWidget):
    def __init__(self, parent=None):
        super(Radiodemo, self).__init__(parent)
        layout = QHBoxLayout()
        self.btn1 = QRadioButton("Button1")
        self.btn1.setChecked(True)
        self.btn1.toggled.connect(lambda:self.btnstate(self.btn1))
        layout.addWidget(self.btn1)

        self.btn2 = QRadioButton("Button2")
        self.btn2.toggled.connect(lambda:self.btnstate(self.btn2))
        layout.addWidget(self.btn2)
        self.setLayout(layout)
        self.setWindowTitle("RadioButton demo")

    def btnstate(self, btn):
        if btn.text()=="Button1":
            if btn.isChecked() == True:
                print( btn.text() + " is selected" )
            else:
                print( btn.text() + " is deselected" )

        if btn.text()=="Button2":
            if btn.isChecked()== True :
                print( btn.text() + " is selected" )
            else:
                print( btn.text() + " is deselected" )

if __name__ == '__main__':
    app = QApplication(sys.argv)
    radioDemo = Radiodemo()
    radioDemo.show()
    sys.exit(app.exec_())

```



图4-25

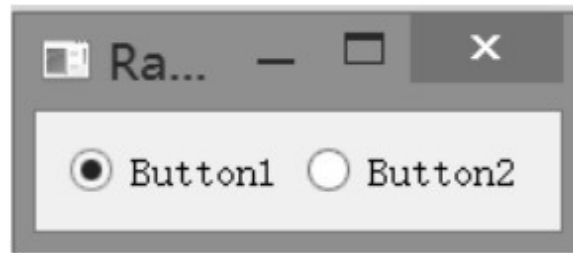


图4-25

```

// 初始化
// 设置初始状态
// 1. 设置 btn1 为初始状态
    self.btn1.setChecked(True)
// 2. 设置 btn2 为初始状态
    toggle 为初始状态
btnstate() 为初始状态
    self.btn1.toggled.connect(lambda: self.btnstate(self.btn
1))
    self.btn2.toggled.connect(lambda: self.btnstate(self.btn
2))
// 3. 设置 toggled 为初始状态
    btnstate() 为初始状态
```

#### 4.5.4 QCheckBox

QCheckBox 是 QAbstractButton 的子类，它继承自 QPushButton。它可以通过 setText() 设置文本，通过 setIcon() 设置图标。QCheckBox 是 QButtonGroup 的子类。

QCheckBox 是 QRadioButton 的子类，它继承自 QAbstractButton。它可以通过 setText() 设置文本，通过 setIcon() 设置图标。QCheckBox 是 QButtonGroup 的子类。

在代码清单4-13中，我们使用QCheckBox类来创建一个复选框。
 首先，我们使用QCheckBox类来创建一个复选框对象。
 然后，我们使用setChecked()方法来设置复选框的状态。
 最后，我们使用isChecked()方法来检查复选框是否被选中。
 在代码清单4-13中，我们使用QCheckBox类来创建一个复选框。
 首先，我们使用QCheckBox类来创建一个复选框对象。
 然后，我们使用setChecked()方法来设置复选框的状态。
 最后，我们使用isChecked()方法来检查复选框是否被选中。
 在代码清单4-13中，我们使用QCheckBox类来创建一个复选框。
 首先，我们使用QCheckBox类来创建一个复选框对象。
 然后，我们使用setChecked()方法来设置复选框的状态。
 最后，我们使用isChecked()方法来检查复选框是否被选中。

图4-13

方 法	描 述
setChecked()	设置复选框的状态，设置为 True 时表示选中复选框，设置为 False 时表示取消选中复选框
setText()	设置复选框的显示文本
text()	返回复选框的显示文本
isChecked()	检查复选框是否被选中
setTriState()	设置复选框为一个三态复选框

图4-14

图4-14

名 称	值	含 义
Qt.Checked	2	组件没有被选中（默认值）
Qt.PartiallyChecked	1	组件被半选中
Qt.Unchecked	0	组件被选中

[图4-16 QCheckBox类](#)

在代码清单4-16中，我们使用PyQt5/Chapter04/qt0410\_QCheckbox.py来创建一个复选框。
 首先，我们使用PyQt5/Chapter04/qt0410\_QCheckbox.py来创建一个复选框。
 然后，我们使用PyQt5/Chapter04/qt0410\_QCheckbox.py来创建一个复选框。
 最后，我们使用PyQt5/Chapter04/qt0410\_QCheckbox.py来创建一个复选框。

```
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from PyQt5.QtCore import Qt

class CheckBoxDemo(QWidget):

    def __init__(self, parent=None):
        super(CheckBoxDemo, self).__init__(parent)

        groupBox = QGroupBox("Checkboxes")
        groupBox.setFlat(True)

        layout = QHBoxLayout()
        self.checkBox1= QCheckBox("&Checkbox1")
        self.checkBox1.setChecked(True)
        self.checkBox1.stateChanged.connect( lambda:self.btnstate
(self.checkBox1) )
```

```

        layout.addWidget(self.checkBox1)

        self.checkBox2 = QCheckBox("Checkbox2")
        self.checkBox2.toggled.connect( lambda:self.btnstate
(self.checkBox2) )
        layout.addWidget(self.checkBox2)

        self.checkBox3 = QCheckBox("Checkbox3")
        self.checkBox3.setTristate(True)
        self.checkBox3.setCheckState(Qt.PartiallyChecked )
        self.checkBox3.stateChanged.connect( lambda:self.btnstate
(self.checkBox3) )
        layout.addWidget(self.checkBox3)

        groupBox.setLayout(layout)
        mainLayout = QVBoxLayout()
        mainLayout.addWidget(groupBox)

        self.setLayout(mainLayout)
        self.setWindowTitle("CheckBox demo")

    def btnstate(self,btn ):
        chk1Status = self.checkBox1.text()+" , isChecked="+
str( self.checkBox1.isChecked() ) + ' , checkState=' +
str(self.checkBox1.checkState())  +"\n"
        chk2Status = self.checkBox2.text()+" , isChecked="+
str( self.checkBox2.isChecked() ) + ' , checkState=' +
str(self.checkBox2.checkState())  +"\n"
        chk3Status = self.checkBox3.text()+" , isChecked="+
str( self.checkBox3.isChecked() ) + ' , checkState=' +
str(self.checkBox3.checkState())  +"\n"
        print(chk1Status + chk2Status + chk3Status )

if __name__ == '__main__':
    app = QApplication(sys.argv)
    checkboxDemo = CheckBoxDemo()
    checkboxDemo.show()
    sys.exit(app.exec_())

```

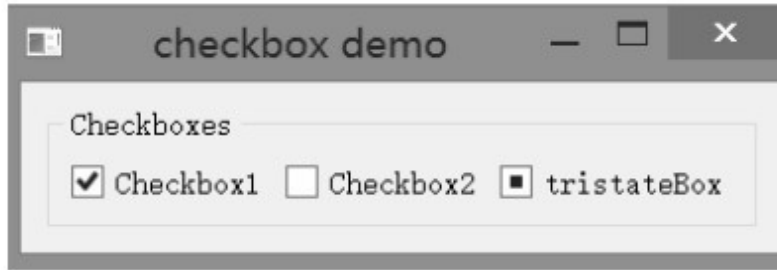


图4-26

```

class MainWindow(QMainWindow)
{
public:
    MainWindow()
    {
        groupBox=QGroupBox("Checkboxes")
        groupBox.setFlat( False )
        connect(checkbox1, &QCheckBox::stateChanged, this, &MainWindow::stateChanged())
        connect(checkbox2, &QCheckBox::stateChanged, this, &MainWindow::stateChanged())
        connect(tristateBox, &QCheckBox::stateChanged, this, &MainWindow::stateChanged())
        btnstate()
    }
    void stateChanged()
    {
        self.checkBox1.stateChanged.connect(
            lambda:self.btnstate(self.checkBox1) )
        self.checkBox2.toggled.connect(
            lambda:self.btnstate(self.checkBox2) )
        self.checkBox3.stateChanged.connect(
            lambda:self.btnstate(self.checkBox3) )
    }
}

```

图4-15

图4-15

控件类型	控件名称	显示的文本	功 能
QCheckBox	checkBox1	Checkbox1	两种状态选择
QCheckBox	checkBox2	Checkbox2	两种状态选择
QCheckBox	checkBox3	tristateBox	三种状态选择

```

        checkBox1.checkBox2.setChecked(checkBox1.isChecked())
        checkBox1.setText("&" + checkBox1.text() + "&Checkbox 1" + "Alt+C")
        checkBox1.setChecked(
            self.checkBox1=QCheckBox("&Checkbox1")
            self.checkBox1.setChecked(True)
            self.checkBox2=QCheckBox("Checkbox2")
            checkBox1.isChecked()
            chk1Status=self.checkBox1.text()+",isChecked="+
            str( self.checkBox1.isChecked() ) + ',checkState=' +
            str(self.checkBox1.checkState()) + "\n"
            self.checkBox=QCheckBox("checkbox3")
            self.checkBox.setTristate()
            self.checkBox3=QCheckBox("Checkbox3")
            self.checkBox3.setTristate(True)
            self.checkBox3.setCheckState(Qt.PartiallyChecked )
            self.checkBox3.stateChanged.connect(
            lambda:self.btnstate(self.checkBox3) )

```

## 4.6 QComboBox

QComboBox 是 Qt 库中提供的一个下拉菜单控件，它允许用户从列表中选择一项。QComboBox 的构造函数如下所示：

方 法	描 述
addItem()	添加一个下拉选项
addItems()	从列表中添加下拉选项
Clear()	删除下拉选项集中的所有选项
count()	返回下拉选项集中的数目
currentText()	返回选中选项的文本
itemText( i )	获取索引为 i 的 item 的选项文本
currentIndex()	返回选中项的索引
setItemText(int index,text)	改变序号为 index 项的文本

## QComboBox 4-17

4-17

信 号	含 义
Activated	当用户选中一个下拉选项时发射该信号
currentIndexChanged	当下拉选项的索引发生改变时发射该信号
highlighted	当选中一个已经选中的下拉选项时，发射该信号

### 4-17 QComboBox

PyQt5/Chapter04/qt0411\_QComboBox.py

PyQt 5 QComboBox

```
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class ComboxDemo(QWidget):
```

```

def __init__(self, parent=None):
    super(ComboxDemo, self).__init__(parent)
    self.setWindowTitle("ComBox 例子")
    self.resize(300, 90)
    layout = QVBoxLayout()
    self.lbl = QLabel("")

    self.cb = QComboBox()
    self.cb.addItem("C")
    self.cb.addItem("C++")
    self.cb.addItems(["Java", "C#", "Python"])
    self.cb.currentIndexChanged.connect(self.selectionchange)
    layout.addWidget(self.cb)
    layout.addWidget(self.lbl)
    self.setLayout(layout)

def selectionchange(self,i):
    self.lbl.setText( self.cb.currentText() )
    print( "Items in the list are :" )
    for count in range(self.cb.count()):
        print( 'item'+str(count) + '=' + self.cb.itemText(count) )
    print( "Current index",i,"selection changed
",self.cb.currentText() )

if __name__ == '__main__':
    app = QApplication(sys.argv)
    comboxDemo = ComboxDemo()
    comboxDemo.show()
    sys.exit(app.exec_())

```

4-27



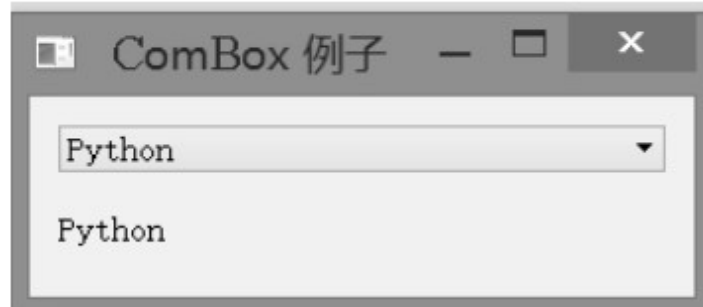


图4-27

```

# 初始化
# 添加5个项目
QComboBox.addItem() 添加单个项目
QComboBox.addItems() 添加多个项目
# 设置当前索引
self.cb=QComboBox()
self.cb.addItem("C")
self.cb.addItem("C++")
self.cb.addItems(["Java","C#","Python"])
# 连接currentIndexChanged信号到selectionchange槽
self.cb.currentIndexChanged.connect(self.selectionchange)

# 定义selectionchange槽函数
def selectionchange(self,i):
    self.lbl.setText( self.cb.currentText() )

```

## 4.7 QSpinBox

QSpinBox 是 Qt 中用于显示和编辑整数的控件。它继承自 QAbstractSpinBox，并实现了 QAbstractSpinBox 的接口。QSpinBox 的默认范围是从 0 到 99，步长为 1。

QSpinBox 是 QDoubleSpinBox 的基类。QAbstractSpinBox 是 QSpinBox 和 QDoubleSpinBox 的基类。QSpinBox 和 QDoubleSpinBox 都实现了 QAbstractSpinBox 的接口。QDoubleSpinBox 是用于显示和编辑浮点数的控件。它继承自 QSpinBox，并实现了 QAbstractSpinBox 的接口。QDoubleSpinBox 的默认范围是从 0.0 到 1.0，步长为 0.1。setDecimals() 用于设置小数位数。

QSpinBox 的默认范围是从 4 到 18。

表 4-18

方 法	描 述
setMinimum()	设置计数器的下界
setMaximum()	设置计数器的上界
setRange()	设置计数器的最大值、最小值和步长值
setValue()	设置计数器的当前值
Value()	返回计数器的当前值
singleStep()	设置计数器的步长值

QSpinBox 的 valueChanged 信号在值改变时发出。可以通过 value() 方法获取当前的值。

### [4-18 QSpinBox](#)

在 PyQt5/Chapter04/qt0412\_QSpinBox.py 文件中，我们使用 PyQt 5 的 QSpinBox 控件。

```

import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class spindemo(QWidget):
    def __init__(self, parent=None):
        super(spindemo, self).__init__(parent)
        self.setWindowTitle("SpinBox 例子")
        self.resize(300, 100)

        layout = QVBoxLayout()
        self.l1=QLabel("current value:")
        self.l1.setAlignment(Qt.AlignCenter)
        layout.addWidget(self.l1)
        self.sp = QSpinBox()
        layout.addWidget(self.sp)
        self.sp.valueChanged.connect(self.valuechange)
        self.setLayout(layout)

    def valuechange(self):
        self.l1.setText("current value:" + str(self.sp.value()) )

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = spindemo()
    ex.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□4-28□□□



方 法	描 述
setMinimum()	设置滑动条控件的最小值
setMaximum()	设置滑动条控件的最大值
setSingleStep()	设置滑动条控件递增/递减的步长值
setValue()	设置滑动条控件的值
value()	获得滑动条控件的值
setTickInterval()	设置刻度间隔
setTickPosition()	设置刻度标记的位置，可以输入一个枚举值，这个枚举值指定刻度线相对于滑块和用户操作的位置。以下是可以输入的枚举值： <ul style="list-style-type: none"><li>• QSlider.NoTicks，不绘制任何刻度线</li><li>• QSlider.TicksBothSides，在滑块的两侧绘制刻度线</li><li>• QSlider.TicksAbove，在（水平）滑块上方绘制刻度线</li><li>• QSlider.TicksBelow，在（水平）滑块下方绘制刻度线</li><li>• QSlider.TicksLeft，在（垂直）滑块左侧绘制刻度线</li><li>• QSlider.TicksRight，在（垂直）滑块右侧绘制刻度线</li></ul>

QSlider 4-20

4-20

信 号	描 述
valueChanged	当滑块的值发生改变时发射此信号。此信号是最常用的
sliderPressed	当用户按下滑块时发射此信号
sliderMoved	当用户拖动滑块时发射此信号
sliderReleased	当用户释放滑块时发射此信号

4-19 QSlider

PyQt5/Chapter04/qt0413\_QSlider.py PyQt 5  
QSlider

```
class SliderDemo(QWidget):
    def __init__(self, parent=None):
        super(SliderDemo, self).__init__(parent)
        self.setWindowTitle("QSlider 例子")
        self.resize(300, 100)

        layout = QVBoxLayout()
        self.l1 = QLabel("Hello PyQt5")
        self.l1.setAlignment(Qt.AlignCenter)
        layout.addWidget(self.l1)
        # 水平方向
        self.sl = QSlider(Qt.Horizontal)
        # 设置最小值
        self.sl.setMinimum(10)
        # 设置最大值
        self.sl.setMaximum(50)
        # 步长
        self.sl.setSingleStep( 3 )
        # 设置当前值
        self.sl.setValue(20)
        # 刻度位置, 刻度在下方
        self.sl.setTickPosition(QSlider.TicksBelow)
        # 设置刻度间隔
        self.sl.setTickInterval(5)
        layout.addWidget(self.sl)
```

```

# 连接信号槽
self.sl.valueChanged.connect(self.valuechange)
self.setLayout(layout)

def valuechange(self):
    print('current slider value=%s' % self.sl.value() )
    size = self.sl.value()
    self.l1.setFont(QFont("Arial",size))

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = SliderDemo()
    demo.show()
    sys.exit(app.exec_())

```

图4-29



图4-29

```

class SliderDemo(QWidget):
    def __init__(self):
        super(SliderDemo, self).__init__()
        self.sl = QSlider(Qt.Horizontal)
        self.sl.valueChanged.connect(self.valuechange)
        self.l1 = QLabel("Hello PyQt5")
        size=self.sl.value()
        self.l1.setFont(QFont("Arial",size))

```

调用 `setTickInterval(5)` 每隔 5 秒更新一次，9 秒后更新 10 次，  
即  $(50-10)/5 + 1 = 9$  次。

```
# 设置范围  
self.sl.setMinimum(10)  
# 设置范围  
self.sl.setMaximum(50)  
# 设置间隔  
self.sl.setTickInterval(5)
```

## 4.9 对话框

### 4.9.1 QDialog

对话框是 Windows 和 Linux 中常见的图形用户界面元素。PyQt 5 提供了 `QDialog` 类，用于创建对话框。

`QDialog` 是 `QMessageBox`、`QFileDialog`、`QFontDialog`、`QInputDialog` 等对话框类的基础类。

`QDialog` 类在 PyQt 5 中的位置如图 4-30 所示。



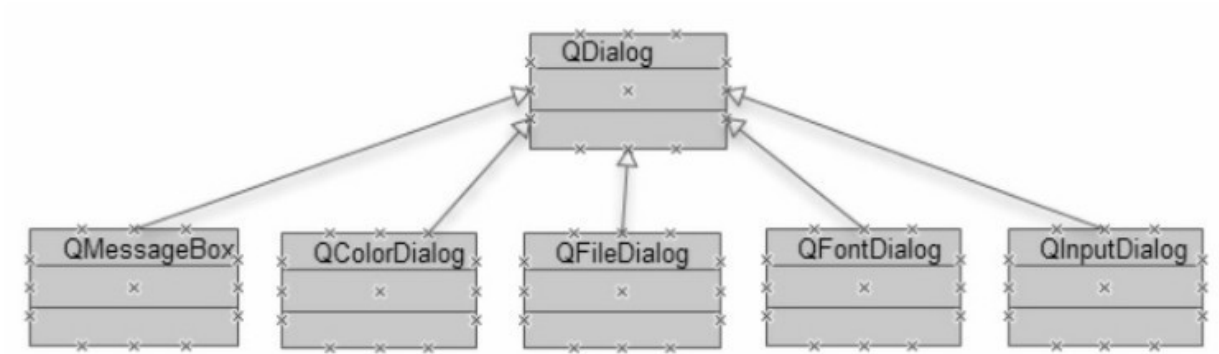


图4-30

QDialog 类的方法如表4-21所示

表4-21

方 法	描 述
setWindowTitle()	设置对话框标题
setWindowModality()	设置窗口模态。取值如下： <ul style="list-style-type: none"> <li>• Qt.NonModal，非模态，可以和程序的其他窗口交互</li> <li>• Qt.WindowModal，窗口模态，程序在未处理完当前对话框时，将阻止和对话框的父窗口进行交互</li> <li>• Qt.ApplicationModal，应用程序模态，阻止和任何其他窗口进行交互</li> </ul>

## 4-20 QDialog 类

本节介绍 PyQt5/Chapter04/qt0416\_Dialog.py 文件中的 PyQt 5 类 QDialog 的相关内容。

```

import sys
from PyQt5.QtCore import *

```

```

from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class DialogDemo( QMainWindow ):

    def __init__(self, parent=None):
        super(DialogDemo, self).__init__(parent)
        self.setWindowTitle("Dialog 例子")
        self.resize(350,300)

        self.btn = QPushButton( self)
        self.btn.setText("弹出对话框")
        self.btn.move(50,50)
        self.btn.clicked.connect(self.showdialog)

    def showdialog(self ):
        dialog = QDialog()
        btn = QPushButton("ok", dialog )
        btn.move(50,50)
        dialog.setWindowTitle("Dialog")
        dialog.setWindowModality(Qt.ApplicationModal)
        dialog.exec_()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = DialogDemo()
    demo.show()
    sys.exit(app.exec_())

```

0000000000004-31000



QMessageBox 类提供了 10 个静态方法，用于显示不同类型的消息对话框。这些方法都继承自 QMessageBox 类。

QMessageBox 类提供了 10 个静态方法，用于显示不同类型的消息对话框。这些方法都继承自 QMessageBox 类。

QMessageBox 类 4-22

4-22

方 法	描 述
information(QWidget parent, title, text, buttons, defaultButton)	弹出消息对话框，各参数解释如下： <ul style="list-style-type: none"><li>• parent，指定的父窗口控件</li><li>• title，对话框标题</li><li>• text，对话框文本</li><li>• buttons：多个标准按钮，默认为 OK 按钮</li><li>• defaultButton：默认选中的标准按钮，默认是第一个标准按钮</li></ul>
question(QWidget parent, title, text, buttons, defaultButton)	弹出问答对话框（各参数解释同上）
warning(QWidget parent, title, text, buttons, defaultButton)	弹出警告对话框（各参数解释同上）
critical(QWidget parent, title, text, buttons, defaultButton)	弹出严重错误对话框（各参数解释同上）
about(QWidget parent, title, text)	弹出关于对话框（各参数解释同上）
setTitle()	设置标题
setText()	设置消息正文
setIcon()	设置弹出对话框的图片

QMessageBox 类 4-23

4-23

类 型	描 述
QMessage.Ok	同意操作
QMessage.Cancel	取消操作
QMessage.Yes	同意操作
QMessage.No	取消操作
QMessage.Abort	终止操作
QMessage.Retry	重试操作
QMessage.Ignore	忽略操作

5 5 4-24

4-24

对话框类型	显示效果
<p>消息对话框，用来告诉用户关于提示消息</p> <p><code>QMessageBox.information(self, "标题", "消息对话框正文", QMessageBox.Yes   QMessageBox.No, QMessageBox.Yes)</code></p>	
<p>提问对话框，用来告诉用户关于提问消息</p> <p><code>QMessageBox.question(self, "标题", "提问框消息正文", QMessageBox.Yes   QMessageBox.No, QMessageBox.Yes)</code></p>	
<p>警告对话框，用来告诉用户关于不寻常的错误消息</p> <p><code>QMessageBox.warning(self, "标题", "警告框消息正文", QMessageBox.Yes   QMessageBox.No, QMessageBox.Yes)</code></p>	
<p>严重错误对话框，用来告诉用户关于严重的错误消息</p> <p><code>QMessageBox.critical(self, "标题", "严重错误对话框消息正文", QMessageBox.Yes   QMessageBox.No, QMessageBox.Yes)</code></p>	
<p>关于对话框</p> <p><code>QMessageBox.about(self, "标题", "关于对话框")</code></p>	

## 4-21 QMessageBox

PyQt5/Chapter04/qt04\_QMessageBox.py PyQt  
5QMessageBox

```

import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class MyWindow( QWidget):
    def __init__(self):
        super(MyWindow,self).__init__()
        self.setWindowTitle("QMessageBox 例子")
        self.resize(300, 100)
        self.myButton = QPushButton(self)
        self.myButton.setText("点击弹出消息框")
        self.myButton.clicked.connect(self.msg)

    def msg(self):
        # 使用 infomation 信息框
        reply = QMessageBox.information(self, "标题", "消息正文",
QMessageBox.Yes | QMessageBox.No , QMessageBox.Yes )
        print( reply )

if __name__ == '__main__':
    app= QApplication(sys.argv)
    myshow=MyWindow()
    myshow.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□4-32□□□

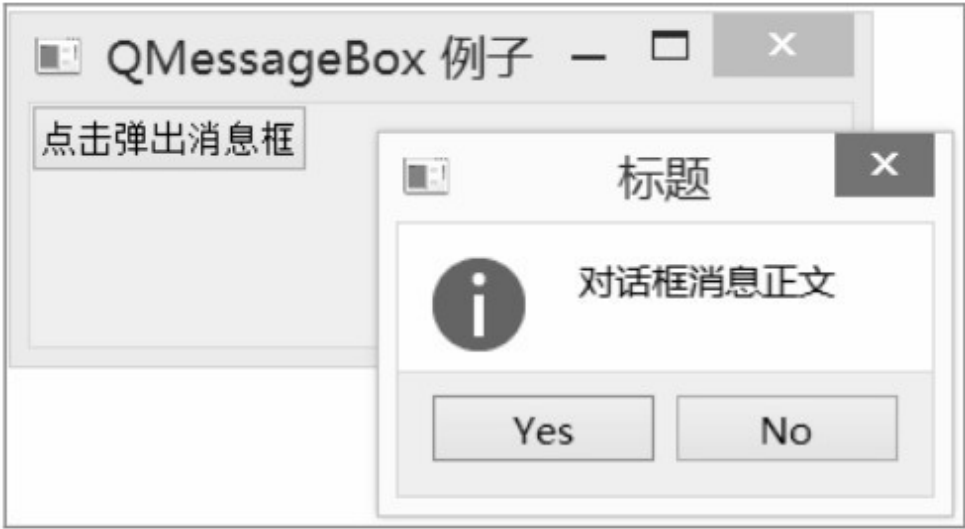


图4-32

### 4.9.3 QInputDialog

QInputDialog 提供了从控件中获取标准整数输入、标准浮点数输入、标准字符串输入和标准列表项输入的方法。它包含两个子类：QInputDialog 和 QInputDialog。QInputDialog 是 QInputDialog 的子类，它提供了从控件中获取标准整数输入、标准浮点数输入、标准字符串输入和标准列表项输入的方法。QInputDialog 是 QInputDialog 的子类，它提供了从控件中获取标准整数输入、标准浮点数输入、标准字符串输入和标准列表项输入的方法。

图4-25

方 法	描 述
getInt()	从控件中获得标准整数输入
getDouble()	从控件中获得标准浮点数输入
getText()	从控件中获得标准字符串输入
getItem()	从控件中获得列表里的选项输入

### 4-22 QInputDialog

在 PyQt5/Chapter04/qt04\_QInputDialog.py 文件中，我们使用 PyQt5 的 QInputDialog 类来创建一个简单的对话框。

```
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class InputdialogDemo(QWidget):
    def __init__(self, parent=None):
        super(InputdialogDemo, self).__init__(parent)
        layout = QFormLayout()
        self.btn1 = QPushButton("获得列表里的选项")
        self.btn1.clicked.connect(self.getItem)
        self.le1 = QLineEdit()
        layout.addRow(self.btn1, self.le1)

        self.btn2 = QPushButton("获得字符串")
        self.btn2.clicked.connect(self.getText)
        self.le2 = QLineEdit()
```



```

        layout.addRow(self.btn2,self.le2)

        self.btn3 = QPushButton("获得整数")
        self.btn3.clicked.connect(self.getInt)
        self.le3 = QLineEdit()
        layout.addRow(self.btn3,self.le3)
        self.setLayout(layout)
        self.setWindowTitle("Input Dialog 例子")

    def getItem(self):
        items = ("C", "C++", "Java", "Python")
        item, ok = QInputDialog.getItem(self, "select input dialog",
        "语言列表", items, 0, False)
        if ok and item:
            self.le1.setText(item)

    def getText(self):
        text, ok = QInputDialog.getText(self, 'Text Input Dialog', '输
入姓名:')
        if ok:
            self.le2.setText(str(text))

    def getInt(self):
        num,ok=QInputDialog.getInt(self,"integer input dualog","输入数
字")
        if ok:
            self.le3.setText(str(num))

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = InputdialogDemo()
    demo.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□4-33□□4-36□□□



图4-33



图4-34



图4-35



图4-36

```

    单击事件
    使用QFormLayout布局对话框中的控件
    单击事件
    self.btn1.clicked.connect(self.getItem)
    self.btn2.clicked.connect(self.getText)
    self.btn3.clicked.connect(self.getInt)
    使用QInputDialog.getItem() 使用QInputDialog 使用QComboBox
    QComboBox 使用QComboBox
    def getItem(self):
        items=("C","C++","Java","Python")
        item,ok=QInputDialog.getItem(self,"select input
        dialog",
        "选择",items,0,False)
        if ok and item:
            self.le1.setText(item)
    使用QInputDialog.getText() 使用QInputDialog
    使用QInputDialog.getInt() 使用
    QInputDialog 使用QSpinBox

```

## 4.9.4 QFontDialog

QFontDialog 是 Qt 提供的一个用于选择字体的对话框。它继承自 QDialog，并实现了 QFontDialog 类。通过调用 QFontDialog 的 getFont() 方法，可以获取用户选择的字体。

### 例 4-23 QFontDialog

本例演示了 PyQt5/Chapter04/qt04\_QFontDialog.py 文件中的 PyQt 5 如何使用 QFontDialog 类。

```
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class FontDialogDemo(QWidget):
    def __init__(self, parent=None):
        super(FontDialogDemo, self).__init__(parent)
        layout = QVBoxLayout()
        self.fontButton = QPushButton("choose font")
        self.fontButton.clicked.connect(self.getFont)
        layout.addWidget(self.fontButton)
        self.fontLineEdit = QLabel("Hello, 测试字体例子")
        layout.addWidget(self.fontLineEdit)
        self.setLayout(layout)
        self.setWindowTitle("Font Dialog 例子")

    def getFont(self):
        font, ok = QFontDialog.getFont()
        if ok:
            self.fontLineEdit.setFont(font)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = FontDialogDemo()
    demo.show()
    sys.exit(app.exec_())
```

图4-37 字体选择对话框

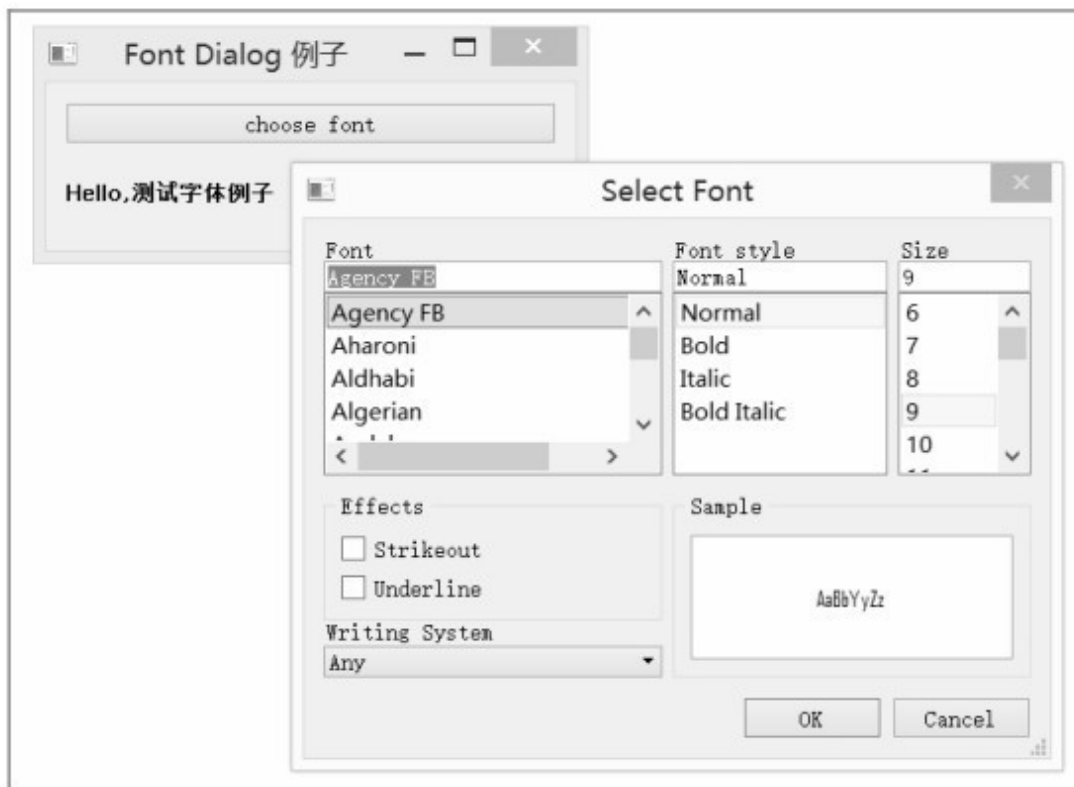


图4-37

代码如下：

在main.cpp文件中添加以下代码，实现点击“choose font”按钮时，弹出字体选择对话框。

在

fontButton.clicked.connect(self.getFont)

添加以下代码：

```
self.btn=QPushButton("choose font")
```

```
self.btn.clicked.connect(self.getFont)
```

```
self.le=QLabel("Hello, 测试字体例子")
```

在fontButton.clicked.connect(self.getFont)中添加以下代码：

fontLineEdit.setText(self.getFont())

```
layout=QVBoxLayout()
```

```

        layout.addWidget(self.btn)
        layout.addWidget(self.le)
        self.fontButton.clicked.connect(self.getFont)
        self.fontButton.clicked.connect(self.getFont)
        self.fontLineEdit.setText(self.getFont())
    def getFont(self):
        font,ok=QFontDialog.getFont()
        if ok:
            self.fontLineEdit.setText(font)

```

## 4.9.5 QFileDialog

QFileDialog 是 Qt 库中的一个类，用于显示文件对话框。QFileDialog 是 QDialog 的子类。

QFileDialog 提供了两个主要的方法：getOpenFileName() 和 getSaveFileName()。getOpenFileName() 用于打开文件，getSaveFileName() 用于保存文件。

QFileDialog 的构造函数如下：

图 4-26

方 法	描 述
getOpenFileName()	返回用户所选择文件的名称，并打开该文件
getSaveFileName()	使用用户选择的文件名并保存文件

方 法	描 述
setFileMode()	可以选择的文件类型，枚举常量是： <ul style="list-style-type: none"> <li>• QFileDialog.AnyFile, 任何文件</li> <li>• QFileDialog.ExistingFile, 已存在的文件</li> <li>• QFileDialog.Directory, 文件目录</li> <li>• QFileDialog.ExistingFiles, 已经存在的多个文件</li> </ul>
setFilter()	设置过滤器，只显示过滤器允许的文件类型

## 4-24 QFileDialog

PyQt5/Chapter04/qt04\_QFileDialog.py PyQt 5  
QFileDialog

```
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class filedialogdemo(QWidget):
    def __init__(self, parent=None):
        super(filedialogdemo, self).__init__(parent)
        layout = QVBoxLayout()
        self.btn = QPushButton("加载图片")
        self.btn.clicked.connect(self.getfile)
        layout.addWidget(self.btn)
        self.le = QLabel("")
        layout.addWidget(self.le)
        self.btn1 = QPushButton("加载文本文件")
        self.btn1.clicked.connect(self.getfiles)
        layout.addWidget(self.btn1)
        self.contents = QTextEdit()
        layout.addWidget(self.contents)
        self.setLayout(layout)
        self.setWindowTitle("File Dialog 例子")

    def getfile(self):
        fname, _ = QFileDialog.getOpenFileName(self, 'Open file',
        'c:\\', "Image files (*.jpg *.gif)")
        self.le.setPixmap(QPixmap(fname))

    def getfiles(self):
```

```

dlg = QFileDialog()
dlg.setFileMode(QFileDialog.AnyFile)
dlg.setFilter( QDir.Files )

if dlg.exec_():
    filenames= dlg.selectedFiles()
    f = open(filenames[0], 'r')

    with f:
        data = f.read()
        self.contents.setText(data)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = filedialogdemo()
    ex.show()
    sys.exit(app.exec_())

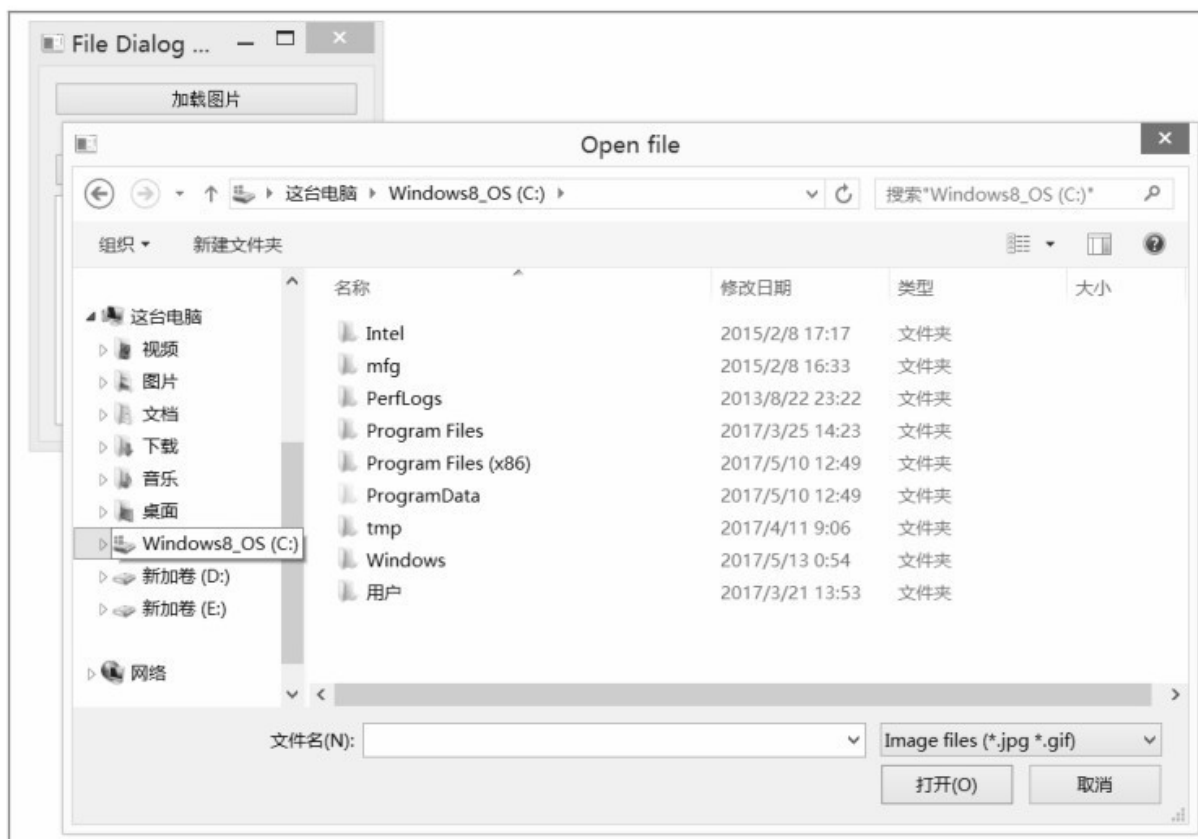
```

□□□□□□□□□□4-38□□4-39□□4-40□□□



□4-38







```

        pixmap = QPixmap(filename)
        self.label.setPixmap(pixmap)
        self.lineEdit.setText(filename)

    def open(self):
        filename, _ = QFileDialog.getOpenFileName(self, 'Open file', 'c:\\', "Image files (*.jpg *.gif)")
        self.le.setPixmap(QPixmap(filename))
        filename, _ = QFileDialog.getOpenFileName(self, 'Open file', 'c:\\', "Image files (*.jpg *.gif)")
        self.le.setPixmap(QPixmap(filename))

    def getfiles(self):
        dlg = QFileDialog()
        dlg.setFileMode(QFileDialog.AnyFile)
        dlg.setFilter(QDir.Files)
        if dlg.exec_():
            filenames = dlg.selectedFiles()
            f = open(filenames[0], 'r')
            with f:

```

```
data=f.read()
self.contents.setText(data)
```

## 4.10 图形用户界面

在本章中，我们将介绍 PyQt5 中的图形用户界面。我们将讨论 QPainter、QPen、QBrush、QPixmap 和 QImage 类。我们将讨论如何使用这些类来创建图形用户界面。我们将讨论如何使用 QPainter 和 QPixmap 类来创建图形用户界面。

### 4.10.1 QPainter

QPainter 是 QWidget 的子类。它用于在 QWidget 上绘制图形。它提供了许多方法，用于绘制各种图形。它提供了许多方法，用于绘制各种图形。

在 PyQt5 中，QWidget.paintEvent() 方法用于处理 QWidget 的绘制事件。它调用了 QPainter 的 begin() 和 end() 方法。QPainter 类提供了许多方法，用于绘制各种图形。图 4-27 显示了 QPainter 类的方法。

图 4-27

方 法	描 述
begin()	开始在目标设备上绘制
drawArc()	在起始角度和最终角度之间画弧
drawEllipse()	在一个矩形内画一个椭圆

方 法	描 述
drawLine(int x1, int y1, int x2, int y2)	绘制一条指定了端点坐标的线。绘制从(x1, y1)到(x2, y2)的直线并且设置当前画笔位置为(x2, y2)
drawPixmap()	从图像文件中提取 Pixmap 并将其显示在指定的位置
drawPolygon()	使用坐标数组绘制多边形
drawRect(int x, int y, int w, int h)	以给定的宽度 w 和高度 h 从左上角坐标(x, y)绘制一个矩形
drawText()	显示给定坐标处的文字
fillRect()	使用 QColor 参数填充矩形
setBrush()	设置画笔风格
setPen()	设置用于绘制的笔的颜色、大小和样式

Qt 的 QPainter 类使用 Qt 的 PenStyle 枚举类型来指定画笔的样式。图 4-28 展示了 Qt 的 PenStyle 枚举类型。图 4-41 展示了 Qt 的 PenStyle 枚举类型的示例。

图 4-28

枚举类型	描 述
Qt.NoPen	没有线。比如 QPainter.drawRect() 填充，但没有绘制任何边界线
Qt.SolidLine	一条简单的线
Qt.DashLine	由一些像素分隔的短线
Qt.DotLine	由一些像素分隔的点
Qt.DashDotLine	轮流交替的点和短线
Qt.DashDotDotLine	一条短线、两个点
Qt.MPenStyle	画笔风格的掩码

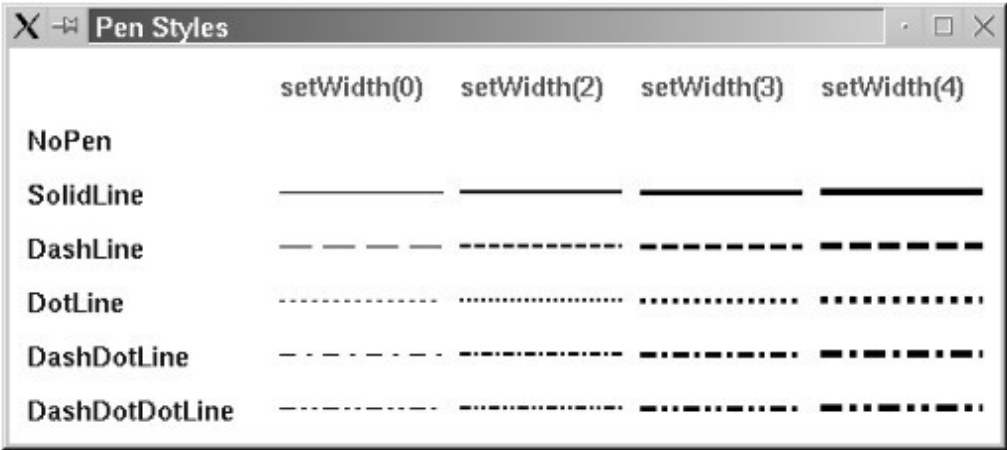


图 4-41

[图 4-25](#)

## PyQt5/Chapter04/qt04\_drawText.py

```
import sys
from PyQt5.QtWidgets import QApplication ,QWidget

from PyQt5.QtGui import QPainter ,QColor ,QFont
from PyQt5.QtCore import Qt

class Drawing(QWidget):
    def __init__(self,parent=None):
        super(Drawing,self).__init__(parent)
        self.setWindowTitle("在窗口中绘制文字")
        self.resize(300, 200)
        self.text = '欢迎学习 PyQt5'

    def paintEvent(self,event):
        painter = QPainter(self)
        painter.begin(self)
        # 自定义绘制方法
        self.drawText(event, painter)
        painter.end()

    def drawText(self, event, qp):
        # 设置画笔的颜色
        qp.setPen( QColor(168, 34, 3) )
        # 设置字体
        qp.setFont( QFont('SimSun', 20))
        # 绘制文字
        qp.drawText(event.rect(), Qt.AlignCenter, self.text)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    demo = Drawing()
    demo.show()
    sys.exit(app.exec_())
```

4-42



□4-42

□ □ □ □ □

[illegible]

```
class Winform(QWidget):
    def __init__(self,parent=None):
        .....
        self.text = '欢迎学习 PyQt5'
```

[illegible]

```
def paintEvent(self, event):
    painter = QPainter(self)
    painter.begin(self)
    # 自定义绘制方法
    self.drawText(event, painter)
    painter.end()
```

```
QtGui.QPainter begin() end()
```

drawText()

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □

```

def drawText(self, event, qp):
    # 设置笔的颜色
    qp.setPen( QColor(168, 34, 3) )
    # 设置字体
    qp.setFont( QFont('SimSun', 20))
    # 绘制文字
    qp.drawText(event.rect(), Qt.AlignCenter, self.text)

```

## 图4-26 点

使用 QPainter 的 drawPoint() 方法可以在指定的位置绘制一个点。使用 QPainter 的 drawPoints() 方法可以在指定的位置绘制多个点。

在 PyQt5/Chapter04/qt04\_drawPoint.py 文件中，使用 QPainter 的 drawPoint() 方法可以在指定的位置绘制一个点。

```

import sys, math
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
from PyQt5.QtCore import Qt

class Drawing(QWidget):
    def __init__(self, parent=None):
        super(Drawing, self).__init__(parent)

```



```

        self.resize(300, 200)
        self.setWindowTitle("在窗口中画点")

    def paintEvent(self, event):
        # 初始化绘图工具
        qp = QPainter()
        # 开始在窗口中绘制
        qp.begin(self)
        # 自定义画点方法
        self.drawPoints(qp)
        # 结束在窗口中绘制
        qp.end()

    def drawPoints(self, qp):
        qp.setPen( Qt.red)
        size = self.size()

        for i in range(1000):
            # 绘制正弦函数图形, 它的周期是[-100, 100]
            x = 100 * (-1+2.0*i/1000)+ size.width()/2.0
            y = -50*math.sin((x - size.width()/2.0)*math.pi/50) +
size.height()/2.0
            qp.drawPoint(x, y)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = Drawing()
    demo.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□4-43□□□□

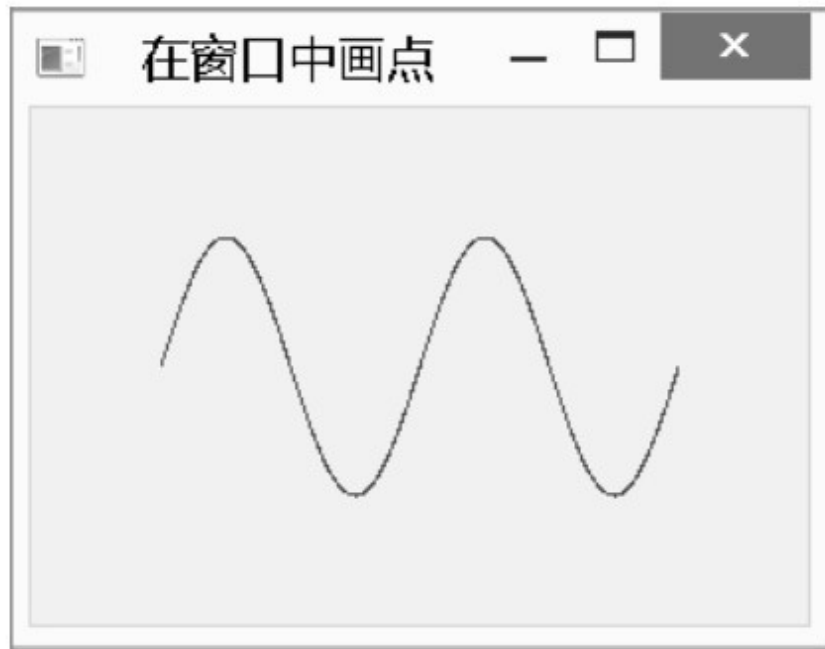


图4-43

```

class Window:
    def __init__(self, size):
        self.size = size
        self.points = []
        for i in range(1000):
            x = 100 * (-1 + 2.0 * i / 1000) + size.width() / 2.0
            y = -50 * math.sin((x - size.width() / 2.0) * math.pi / 50)
            self.points.append((x, y))
        self.draw()
    def draw(self):
        qp = QPainter()
        qp.begin(self)
        qp.setPen(QPen(Qt.red, 1))
        qp.drawPoints(self.points)
        self.update()
        self.show()
        self.size = self.size()
        self.draw()

```

qp.drawPoint(x,y)

## 4.10.2 QPen

QPen 클래스는 선의 색, 두께, 스타일 등을 지정하는 데 사용됩니다.   
예를 들어, 선을 그릴 때 QPen 객체를 사용하여 색과 두께를 지정할 수 있습니다.

### 예제 4-27 QPen 사용하기

이 예제는 PyQt5/Chapter04/qt04\_drawPen.py 파일을 사용하여 QPen 클래스를 사용하여 선을 그리는 방법을 보여줍니다.

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
from PyQt5.QtCore import Qt

class Drawing(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()
```

```
def initUI(self):
    self.setGeometry(300, 300, 280, 270)
    self.setWindowTitle('钢笔样式例子')

def paintEvent(self, e):

    qp = QPainter()
    qp.begin(self)
    self.drawLines(qp)
    qp.end()

def drawLines(self, qp):
    pen = QPen(Qt.black, 2, Qt.SolidLine)

    qp.setPen(pen)
    qp.drawLine(20, 40, 250, 40)

    pen.setStyle(Qt.DashLine)
    qp.setPen(pen)
    qp.drawLine(20, 80, 250, 80)

    pen.setStyle(Qt.DashDotLine)
    qp.setPen(pen)
    qp.drawLine(20, 120, 250, 120)

    pen.setStyle(Qt.DotLine)
    qp.setPen(pen)
    qp.drawLine(20, 160, 250, 160)

    pen.setStyle(Qt.DashDotDotLine)
    qp.setPen(pen)
    qp.drawLine(20, 200, 250, 200)

    pen.setStyle(Qt.CustomDashLine)
    pen.setDashPattern([1, 4, 5, 4])
    qp.setPen(pen)
    qp.drawLine(20, 240, 250, 240)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = Drawing()
```

```
demo.show()
sys.exit(app.exec_())
```

图4-44



图4-44

图4-44

图4-44展示了不同笔刷样式的示例。图中包含6条水平线，其中1条为实线，5条为虚线。这些线条分别由不同的笔刷（QPen）绘制而成，展示了不同的线宽和线型。

图4-44展示了不同笔刷样式的示例。图中包含6条水平线，其中1条为实线，5条为虚线。这些线条分别由不同的笔刷（QPen）绘制而成，展示了不同的线宽和线型。

图4-44展示了不同笔刷样式的示例。图中包含6条水平线，其中1条为实线，5条为虚线。这些线条分别由不同的笔刷（QPen）绘制而成，展示了不同的线宽和线型。

图4-44展示了不同笔刷样式的示例。图中包含6条水平线，其中1条为实线，5条为虚线。这些线条分别由不同的笔刷（QPen）绘制而成，展示了不同的线宽和线型。

图4-44展示了不同笔刷样式的示例。图中包含6条水平线，其中1条为实线，5条为虚线。这些线条分别由不同的笔刷（QPen）绘制而成，展示了不同的线宽和线型。

图4-44展示了不同笔刷样式的示例。图中包含6条水平线，其中1条为实线，5条为虚线。这些线条分别由不同的笔刷（QPen）绘制而成，展示了不同的线宽和线型。

```
pen.setStyle(Qt.CustomDashLine)
```

```
pen.setDashPattern([1,4,5,4])
qp.setPen(pen)
qp.drawLine(20,240,250,240)
```

### 4.10.3 QBrush

QBrush 클래스는 다양한 패턴과 색상으로 도면을 칠할 수 있는 클래스입니다. QBrush 클래스는 QColor 클래스와 함께 사용됩니다. QBrush 클래스의 생성자는 다음과 같습니다.

#### 4-28 QBrush

이 코드는 PyQt5/Chapter04/qt04\_drawBrush.py 파일에 있는 코드입니다. QBrush 클래스를 사용하여 도면을 칠하는 방법을 보여줍니다.

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
from PyQt5.QtCore import Qt

class Drawing(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setGeometry(300, 300, 365, 280)
        self.setWindowTitle('画刷例子')
        self.show()

    def paintEvent(self, e):
        qp = QPainter()
        qp.begin(self)
        self.drawLines(qp)
        qp.end()

    def drawLines(self, qp):
        brush = QBrush(Qt.SolidPattern)
        qp.setBrush(brush)
        qp.drawRect(10, 15, 90, 60)

        brush = QBrush(Qt.Dense1Pattern)
        qp.setBrush(brush)
        qp.drawRect(130, 15, 90, 60)

        brush = QBrush(Qt.Dense2Pattern)
        qp.setBrush(brush)
        qp.drawRect(250, 15, 90, 60)

        brush = QBrush(Qt.Dense3Pattern)
        qp.setBrush(brush)
        qp.drawRect(10, 105, 90, 60)
```

```

brush = QBrush(Qt.DiagCrossPattern)
qp.setBrush(brush)
qp.drawRect(10, 105, 90, 60)

brush = QBrush(Qt.Dense5Pattern)
qp.setBrush(brush)
qp.drawRect(130, 105, 90, 60)

brush = QBrush(Qt.Dense6Pattern)
qp.setBrush(brush)
qp.drawRect(250, 105, 90, 60)

brush = QBrush(Qt.HorPattern)
qp.setBrush(brush)
qp.drawRect(10, 195, 90, 60)

brush = QBrush(Qt.VerPattern)
qp.setBrush(brush)
qp.drawRect(130, 195, 90, 60)

brush = QBrush(Qt.BDiagPattern)
qp.setBrush(brush)
qp.drawRect(250, 195, 90, 60)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = Drawing()
    demo.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□4-45□□□



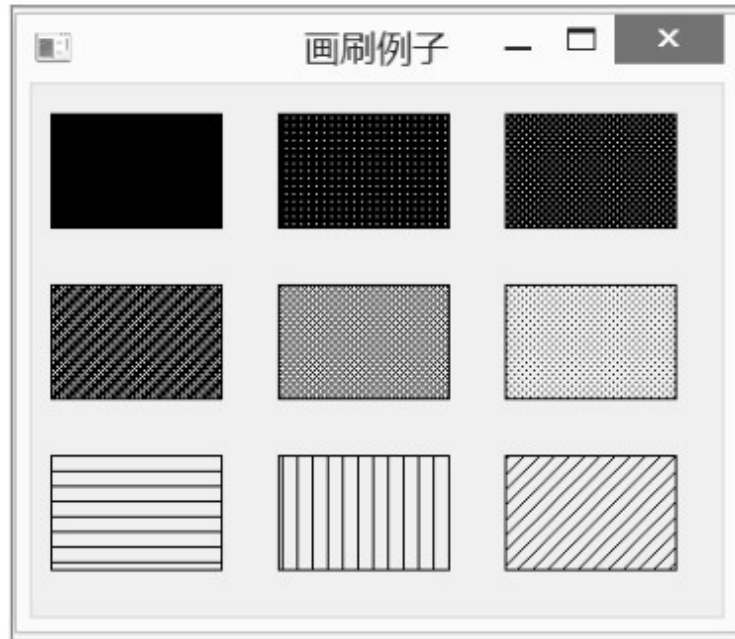


图4-45

```

#include <QtGui>
#include <QPainter>
#include <QBrush>
#include <QRect>

int main()
{
    QPainter qp;
    QBrush brush(Qt::SolidPattern);
    qp.setBrush(brush);
    qp.drawRect(10,15,90,60)
}

```

#### 4.10.4 QPixmap

QPixmap 类用于在 QPainter 类中绘制位图。QPixmap 类是 QImage 类的派生类，它提供了与 QImage 类相同的接口，但 QPixmap 类只支持位图格式。QPixmap 类支持以下格式：BMP、GIF、JPG、JPEG、PNG、PBM、PGM、PPM、XBM、XPM。

QPixmap 类在头文件 `QPixmap.h` 中定义，在库文件 `libqimageio` 中实现。图 4-29 显示了 QPixmap 类的构造函数。

图4-29

方 法	描 述
copy()	从 QRect 对象复制到 QPixmap 对象
fromImage()	将 QImage 对象转换为 QPixmap 对象
grabWidget()	从给定的窗口小控件创建一个像素图
grabWindow()	在窗口中创建数据的像素图
load()	加载图像文件作为 QPixmap 对象
save()	将 QPixmap 对象保存为文件
toImage()	将 QPixmap 对象转换为 QImage 对象

## 图4-29 QPixmap

该程序位于 PyQt5/Chapter04/qt04\_QPixmap.py，该程序是 PyQt 5 中 QPixmap 的一个简单示例。

```
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
if __name__ == '__main__':
    app=QApplication(sys.argv)
    win=QWidget()
    lab1=QLabel()
    lab1.setPixmap(QPixmap("./images/python.jpg"))
    vbox=QVBoxLayout()
    vbox.addWidget(lab1)
    win.setLayout(vbox)
    win.setWindowTitle("QPixmap 示例")
    win.show()
    sys.exit(app.exec_())
```

该程序运行后的效果如图4-46所示。



图4-46

代码如下

通过调用setPixmap()方法将QPixmap对象设置到QLabel对象

```
lab1=QLabel()
```

```
lab1.setPixmap(QPixmap("./images/python.jpg"))
```

## 4.11 拖拽与释放

### 4.11.1 Drag与Drop

在Qt中，拖拽与释放操作是通过QDrag和QMimeData类来实现的。

通过调用MIME类的方法，可以创建QDrag对象和QMimeData对象。

通过MIME类的方法

可以创建MIME对象

MIME (Multipurpose Internet Mail Extension) 是 Internet 邮件系统的一个标准，它定义了如何封装和传输各种类型的文件，如文本、图像、声音、视频等。MIME 允许在电子邮件中嵌入各种类型的文件，使得电子邮件可以包含丰富的多媒体内容。

MIME 定义了如何封装和传输各种类型的文件，如文本、图像、声音、视频等。MIME 允许在电子邮件中嵌入各种类型的文件，使得电子邮件可以包含丰富的多媒体内容。

MIME 定义了如何封装和传输各种类型的文件，如文本、图像、声音、视频等。

- HTML 文件 .html text/html
- XML 文件 .xml text/xml
- XHTML 文件 .xhtml application/xhtml+xml
- 文本文件 .txt text/plain
- RTF 文件 .rtf application/rtf
- PDF 文件 .pdf application/pdf
- Microsoft Word 文件 .word application/msword
- PNG 文件 .png image/png
- GIF 文件 .gif image/gif
- JPEG 文件 .jpeg/.jpg image/jpeg
- au 文件 .au audio/basic
- MIDI 文件 .mid/.midi audio/midi, audio/x-midi
- RealAudio 文件 .ra/.ram audio/x-pn-realaudio
- MPEG 文件 .mpg/.mpeg video/mpeg
- AVI 文件 .avi video/x-msvideo
- GZIP 文件 .gz application/x-gzip
- TAR 文件 .tar application/x-tar
- 任意文件 application/octet-stream

MIME 定义了如何封装和传输各种类型的文件，如文本、图像、声音、视频等。MIME 允许在电子邮件中嵌入各种类型的文件，使得电子邮件可以包含丰富的多媒体内容。

4-30 MimeData 是 MIME 文件

图4-30

判断函数	设置函数	获取函数	MIME 类型
hasText()	text()	setText()	text/plain
hasHtml()	html()	setHtml()	text/html
hasUrls()	urls()	setUrls()	text/uri-list
hasImage()	imageData()	setImageData()	image/*
hasColor()	colorData()	setColorData()	application/x-color

图4-30 QWidget 的拖曳操作函数  
QWidget.setDragEnabled() 方法用于设置控件是否支持拖曳操作。如果调用该函数并传入 True 参数，则控件将支持拖曳操作。图4-31展示了拖曳操作的事件。

图4-31

事 件	描 述
DragEnterEvent	当执行一个拖曳控件操作，并且鼠标指针进入该控件时，这个事件将被触发。在这个事件中可以获得被操作的窗口控件，还可以有条件地接受或拒绝该拖曳操作
DragMoveEvent	在拖曳操作进行时会触发该事件
DragLeaveEvent	当执行一个拖曳控件操作，并且鼠标指针离开该控件时，这个事件将被触发
DropEvent	当拖曳操作在目标控件上被释放时，这个事件将被触发

[图4-30 代码](#)

图4-30展示了PyQt5/Chapter04/qt04\_drag.py文件中PyQt 5的拖曳操作代码。

```
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class Combo(QComboBox):

    def __init__(self, title, parent):
        super(Combo, self).__init__(parent)
        self.setAcceptDrops(True)

    def dragEnterEvent(self, e):
        print(e)
        if e.mimeData().hasText():
            e.accept()
        else:
            e.ignore()

    def dropEvent(self, e):
        self.addItem(e.mimeData().text())

class Example(QWidget):
    def __init__(self):
```



```

        e.accept()
    else:
        e.ignore()

```

## 4.11.2 QClipboard

QClipboard 类用于管理剪贴板内容。它提供了以下方法：

- QDrag 类用于拖拽操作。
- QApplication 类提供了 `clipboard()` 方法，用于获取或设置剪贴板。
- MimeData 类用于处理 MIME 数据。
- QClipboard 类提供了 `clear()`、`setImage()`、`setMimeData()`、`setPixmap()`、`setText()` 和 `text()` 等方法。

表 4-32

方 法	描 述
<code>clear()</code>	清除剪贴板的内容
<code>setImage()</code>	将 QImage 对象复制到剪贴板中
<code>setMimeData()</code>	将 MIME 数据设置为剪贴板
<code>setPixmap()</code>	从剪贴板中复制 QPixmap 对象
<code>setText()</code>	从剪贴板中复制文本
<code>text()</code>	从剪贴板中检索文本

QClipboard 类提供了 `dataChanged()` 信号。

表 4-33

信 号	含 义
<code>dataChanged</code>	当剪贴板内容发生变化时，这个信号被发射

## 4-31 QClipboard 类

以下代码示例展示了如何在 PyQt5 中使用 QClipboard 类。文件名为 `Chapter04/qt04_QClipboard.py`。



```
import os
import sys
from PyQt5.QtCore import QMimeData
from PyQt5.QtWidgets import (QApplication, QDialog, QGridLayout,
QLabel, QPushButton)
from PyQt5.QtGui import QPixmap

class Form(QDialog):
    def __init__(self, parent=None):
        super(Form, self).__init__(parent)
        textCopyButton = QPushButton("&Copy Text")
        textPasteButton = QPushButton("Paste &Text")
        htmlCopyButton = QPushButton("C&opy HTML")
        htmlPasteButton = QPushButton("Paste &HTML")
        imageCopyButton = QPushButton("Co&py Image")
```

```

        imagePasteButton = QPushButton("Paste &Image")
        self.textLabel = QLabel("Original text")
        self.imageLabel = QLabel()
        self.imageLabel.setPixmap(QPixmap(os.path.join(
            os.path.dirname(__file__), "images/clock.png")))
        layout = QGridLayout()
        layout.addWidget(textCopyButton, 0, 0)
        layout.addWidget(imageCopyButton, 0, 1)
        layout.addWidget(htmlCopyButton, 0, 2)
        layout.addWidget(textPasteButton, 1, 0)
        layout.addWidget(imagePasteButton, 1, 1)
        layout.addWidget(htmlPasteButton, 1, 2)
        layout.addWidget(self.textLabel, 2, 0, 1, 2)
        layout.addWidget(self.imageLabel, 2, 2)
        self.setLayout(layout)
        textCopyButton.clicked.connect(self.copyText)
        textPasteButton.clicked.connect(self.pasteText)
        htmlCopyButton.clicked.connect(self.copyHtml)
        htmlPasteButton.clicked.connect(self.pasteHtml)
        imageCopyButton.clicked.connect(self.copyImage)
        imagePasteButton.clicked.connect(self.pasteImage)
        self.setWindowTitle("Clipboard 例子")

    def copyText(self):
        clipboard = QApplication.clipboard()
        clipboard.setText("I've been clipped!")

    def pasteText(self):
        clipboard = QApplication.clipboard()
        self.textLabel.setText(clipboard.text())

    def copyImage(self):
        clipboard = QApplication.clipboard()
        clipboard.setPixmap(QPixmap(os.path.join(
            os.path.dirname(__file__), "../images/python.png")))

    def pasteImage(self):
        clipboard = QApplication.clipboard()
        self.imageLabel.setPixmap(clipboard.pixmap())

    def copyHtml(self):
        mimeTypeData = QMimeData()

```

```

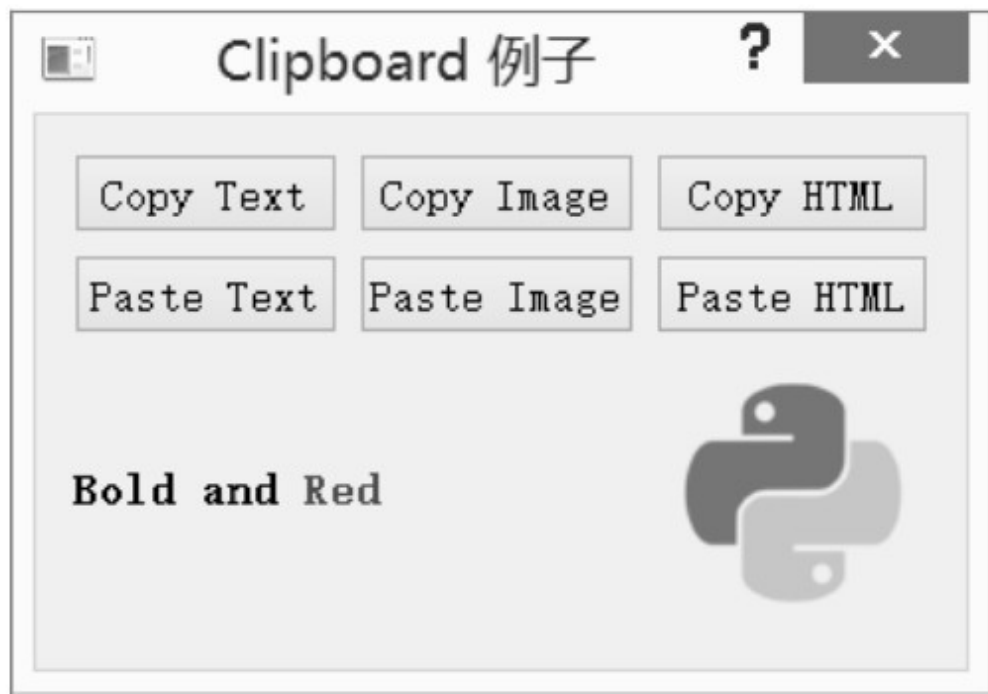
        mimeTypeData.setHtml("<b>Bold and <font color=red>Red</font></b>")
        clipboard = QApplication.clipboard()
        clipboard.setMimeData(mimeTypeData)

    def pasteHtml(self):
        clipboard = QApplication.clipboard()
        mimeTypeData = clipboard.mimeData()
        if mimeTypeData.hasHtml():
            self.textLabel.setText(mimeTypeData.html())

if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = Form()
    form.show()
    sys.exit(app.exec_())

```

例 4-48



例 4-48

例 4-48

```

        clipboard=QApplication.clipboard()
        self.textLabel.setText(clipboard.text())
    def paste(self):
        clipboard=QApplication.clipboard()
        self.imageLabel.setPixmap(clipboard.pixmap())

```

## 4.12 日期时间

### 4.12.1 QCalendar

```

QCalendar 类提供了对日历的访问。它提供了对日历的访问。
QCalendar 类提供了对日历的访问。它提供了对日历的访问。
QCalendar 类提供了对日历的访问。它提供了对日历的访问。

```

方 法	描 述
setDateRange()	设置日期范围供选择
setFirstDayOfWeek()	重新设置星期的第一天，默认是星期日。其参数枚举值如下： <ul style="list-style-type: none"><li>• Qt.Monday，星期一</li><li>• Qt.Tuesday，星期二</li><li>• Qt.Wednesday，星期三</li><li>• Qt.Thursday，星期四</li><li>• Qt.Friday，星期五</li><li>• Qt.Saturday，星期六</li><li>• Qt.Sunday，星期日</li></ul>
setMinimumDate()	设置最大日期
setMaximumDate ()	设置最小日期
setSelectedDate()	设置一个 QDate 对象，作为日期控件所选定的日期
maximumDate	获取日历控件的最大日期
minimumDate	获取日历控件的最小日期
selectedDate()	返回当前选定的日期
setGridvisible ()	设置日历控件是否显示网格

## [4-32 QCalendar](#)

在 PyQt5/Chapter04/qt04\_QCalendar.py 中 PyQt 5 使用 QCalendar 类。

```
import sys
from PyQt5 import QtCore
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from PyQt5.QtCore import QDate
```

```

class CalendarExample( QWidget):
    def __init__(self):
        super(CalendarExample, self).__init__()
        self.initUI()

    def initUI(self):
        self.cal = QCalendarWidget(self)
        self.cal.setMinimumDate(QDate(1980, 1, 1))
        self.cal.setMaximumDate(QDate(3000, 1, 1))
        self.cal.setGridVisible(True)
        self.cal.move(20, 20)
        self.cal.clicked[QtCore.QDate].connect(self.showDate)
        self.lbl = QLabel(self)
        date = self.cal.selectedDate()
        self.lbl.setText(date.toString("yyyy-MM-dd dddd"))
        self.lbl.move(20, 300)
        self.setGeometry(100,100,400,350)
        self.setWindowTitle('Calendar 例子')

    def showDate(self, date):
        self.lbl.setText(date.toString("yyyy-MM-dd dddd") )

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = CalendarExample()
    demo.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□4-49□□□



图4-49

代码如下

新建一个工程，添加头文件，并包含头文件

在main.cpp文件中添加以下代码，并编译运行

```
self.cal=QCalendarWidget(self)
```

```
self.cal.setMinimumDate(QDate(1980,1,1))
```

```
self.cal.setMaximumDate(QDate(3000,1,1))
```

在main.cpp文件中添加以下代码，并编译运行

```
showDate()
```

```
self.cal.clicked[QtCore.QDate].connect(self.showDate)
```

在main.cpp文件中添加以下代码，并编译运行

```
selectedDate()
```

```
def showDate(self,date):
    self.lbl.setText(date.toString("yyyy-MM-dd dddd") )
```

## 4.12.2 QDateTimeEdit

QDateTimeEdit 是 Qt 提供的一个日期和时间编辑控件。它继承自 QLineEdit。QDateTimeEdit 提供了两种日期和时间输入格式：日期和时间（例如 2017-07-16 21:04:56）和日期（例如 2017-07-16）。图 4-50 和图 4-51 展示了 QDateTimeEdit 的两种输入格式。

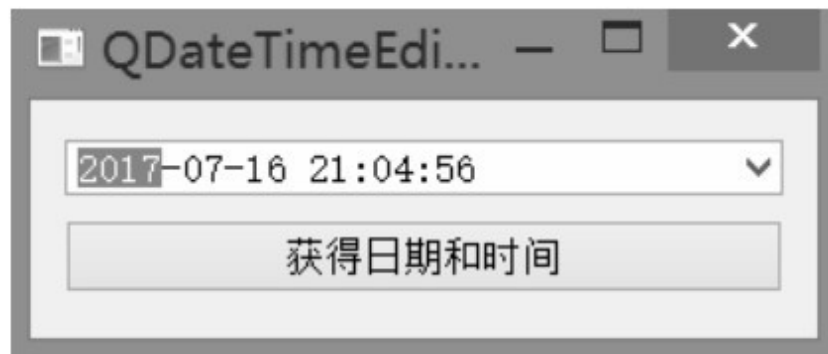


图 4-50

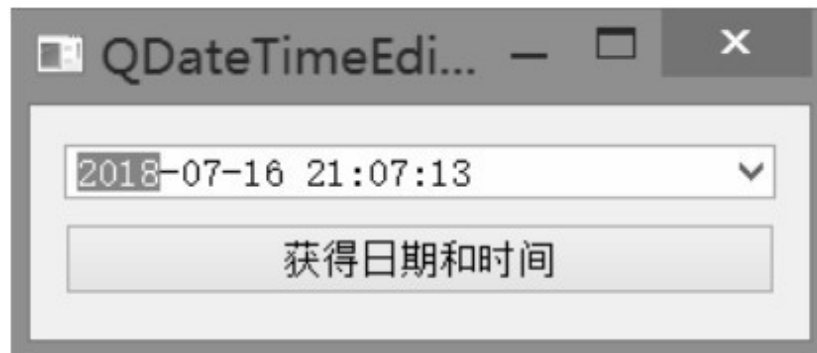


图 4-51

QDateTimeEdit 提供了 setDisplayFormat() 方法，用于设置日期和时间的显示格式。图 4-35 展示了 QDateTimeEdit 的另一种输入格式。

图 4-35



方 法	描 述
setDisplayFormat()	设置日期时间格式： <ul style="list-style-type: none"><li>• yyyy, 代表年份，用 4 位数表示</li><li>• MM, 代表月份，取值范围为 01~12</li><li>• dd, 代表日，取值范围为 01~31</li><li>• HH, 代表小时，取值范围为 00~23</li><li>• mm, 代表分钟，取值范围为 00~59</li><li>• ss, 代表秒，取值范围为 00~59</li></ul>
setMinimumDate()	设置控件的最小日期
setMaximumDate()	设置控件的最大日期
time()	返回编辑的时间
date()	返回编辑的日期

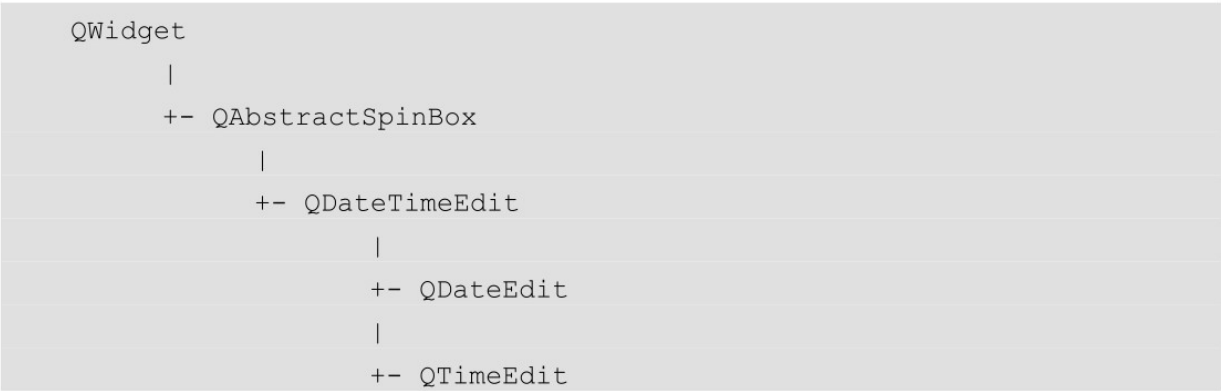
## QDateTimeEdit4-36

4-36

信 号	含 义
dateChanged	当日期改变时发射此信号
dateTimeChanged	当日期时间改变时发射此信号
timeChanged	当时间改变时发射此信号

### 1.QDateTimeEdit

QDateEdit、QTimeEdit和QDateTimeEdit是Qt提供的一个日期时间编辑控件。QDateEdit用于编辑日期，QTimeEdit用于编辑时间，QDateTimeEdit用于编辑日期时间。



QDateEdit、QTimeEdit和QDateTimeEdit是Qt提供的一个日期时间编辑控件。QDateEdit用于编辑日期，QTimeEdit用于编辑时间，QDateTimeEdit用于编辑日期时间。

```
class QDateTimeEdit(QWidget):
    def __init__(self, parent=None):
        QWidget.__init__(self, parent)
        self.dateEdit = QDateEdit(self)
        self.timeEdit = QTimeEdit(self)
```

```
        self.dateEdit.setDisplayFormat("yyyy-MM-dd")
```

```
        self.timeEdit.setDisplayFormat("HH:mm:ss")
```

```
        self.dateEdit.setCalendarPopup(True)
```

```
        self.timeEdit.setCalendarPopup(True)
```

```
class QDateTimeEdit(QWidget):
    def __init__(self, parent=None):
        QWidget.__init__(self, parent)
        self.dateEdit = QDateEdit(self)
        self.timeEdit = QTimeEdit(self)
```

```
        self.dateTimeEdit = QDateTimeEdit(self)
```

```
        self.dateEdit.setCalendarPopup(True)
```

```
        self.timeEdit.setCalendarPopup(True)
```

```
        self.dateTimeEdit.setCalendarPopup(True)
```

## 2. QDateTimeEdit

```
class QDateTimeEdit(QWidget):
    def __init__(self, parent=None):
        QWidget.__init__(self, parent)
        self.dateTimeEdit = QDateTimeEdit(self)
```

```
class QDateTimeEdit(QWidget):
    def __init__(self, parent=None):
        QWidget.__init__(self, parent)
        self.dateTimeEdit = QDateTimeEdit(self)
```

```
        self.dateTimeEdit.setCalendarPopup(True)
```

```
        self.dateTimeEdit2 = QDateTimeEdit(QDateTime.currentDateTime(), self)
```

```
        self.dateEdit = QDateTimeEdit(QDate.currentDate(), self)
```

```
        self.timeEdit = QDateTimeEdit(QTime.currentTime(), self)
```

PyQt5/Chapter04/qt04\_QDateTimeEdit01.py

4-52

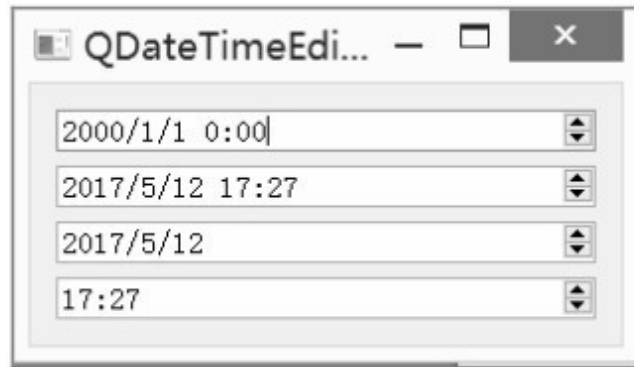


图4-52

QDateTimeEdit 提供了 setDate() 和 setTime() 方法

### 3. 日期时间编辑

```

setDateFormat() 方法用于设置日期时间编辑的显示格式。
dateTimeEdit=QDateTimeEdit(self)
dateTimeEdit2=QDateTimeEdit(QDateTime.currentDate()
eTime(),self)
dateEdit=QDateTimeEdit(QDate.currentDate(),self)
timeEdit=QDateTimeEdit(QTime.currentTime(),self)
# 设置日期时间编辑的显示格式
dateTimeEdit.setDateFormat("yyyy-MM-dd
HH:mm:ss")
dateTimeEdit2.setDateFormat("yyyy/MM/dd HH-mm-
ss")
dateEdit.setDateFormat("yyyy.MM.dd")
timeEdit.setDateFormat("HH:mm:ss")

```

图4-53

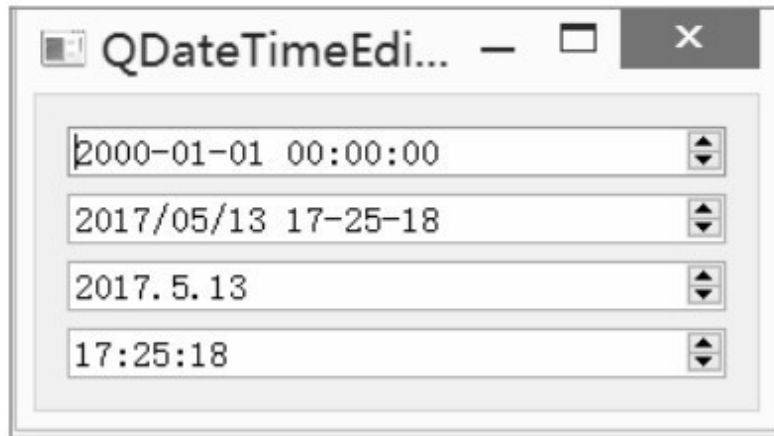


图4-53

#### 4. 日期时间编辑

在代码清单4-53中，我们使用 `QDateTimeEdit` 类来创建一个日期时间编辑框。我们使用 `currentDate` 方法来获取当前的日期，并设置日期编辑框的日期为当前日期。

```
dateEdit=QDateTimeEdit(QDateTime.currentDateTime(),self)
```

```
dateEdit.setDisplayFormat("yyyy-MM-dd HH:mm:ss")
```

```
# 设置日期范围
```

```
dateEdit.setMinimumDate(QDate.currentDate().addDays(-365))
```

```
# 设置日期范围
```

```
dateEdit.setMaximumDate(QDate.currentDate().addDays(365))
```

#### 5. 日历

在代码清单4-54中，我们使用 `QCalendarWidget` 类来创建一个日历。我们使用 `setCalendarPopup` 方法来设置日历的弹出式窗口。

```
dateEdit=QDateTimeEdit(QDateTime.currentDateTime(),self)
```

```
dateEdit.setMinimumDate(QDate.currentDate().addDays(-365))
```

```
dateEdit.setMaximumDate(QDate.currentDate().addDays(365))
```

```
dateEdit.setCalendarPopup( True)
```

图4-54

图4-54

图4-54



图4-54

## 6. 图4-54

图4-54 date() QDateTime() 图4-54

图4-54 QDate year() month() day() 图4-54

```
dateTime=self.dateEdit.dateTime()
```

```

# 日期
maxDate=self.dateEdit.maximumDate()
# 日期时间
maxDateTime=self.dateEdit.maximumDateTime()
# 时间
maxTime=self.dateEdit.maximumTime()
# 日期
minDate=self.dateEdit.minimumDate()
# 日期时间
minDateTime=self.dateEdit.minimumDateTime()
# 时间
minTime=self.dateEdit.minimumTime()
print('\n日期时间' )
print('dateTime=%s' % str(dateTime) )
print('maxDate=%s' % str(maxDate) )
print('maxDateTime=%s' % str(maxDateTime) )
print('maxTime=%s' % str(maxTime) )
print('minDate=%s' % str(minDate) )
print('minDateTime=%s' % str(minDateTime) )
print('minTime=%s' % str(minTime) )

# 日期时间
dateTime=PyQt5.QtCore.QDateTime(2017,5,1,17,41,2
2,441)
maxDate=PyQt5.QtCore.QDate(2018,5,13)
maxDateTime=PyQt5.QtCore.QDateTime(2018,5,13,23
,59,59,999)

```

```

maxTime=PyQt5.QtCore.QTime(23,59,59,999)
minDate=PyQt5.QtCore.QDate(2016,5,13)
minDateTime=PyQt5.QtCore.QDateTime(2016,5,13,0,0
)
minTime=PyQt5.QtCore.QTime(0,0)

```

## 7. 日期时间编辑

```

QDateTimeEdit  日期时间编辑  dateChanged  日期时间
dateTimeChanged  时间变化  timeChanged  时间变化
日期时间变化  日期时间变化
日期时间变化  日期时间变化
dateEdit.dateChanged.connect(self.onDateChanged)
dateEdit.dateTimeChanged.connect(self.onDateTimeCh
anged)
dateEdit.timeChanged.connect(self.onTimeChanged)
日期时间
# 日期时间变化
def onDateChanged(self ,date):
    print(date)
# 日期时间变化
def onDateTimeChanged(self ,dateTime ):
    print(dateTime)
# 时间变化
def onTimeChanged(self ,time):
    print(time)

```

## 4-33 QDateTimeEdit

PyQt5/Chapter04/qt04\_QDateTimeEdit02.py
 PyQt 5 QDateTimeEdit

```
import sys
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from PyQt5.QtCore import QDate, QDateTime , QTime

class DateTimeEditDemo(QWidget):
    def __init__(self):
```



```

        super(DateTimeEditDemo, self).__init__()
        self.initUI()

    def initUI(self):
        self.setWindowTitle('QDateTimeEdit 例子')
        self.resize(300, 90)

        vlayout = QVBoxLayout()
        self.dateEdit = QDateTimeEdit(QDateTime.currentDateTime(),
self)

        self.dateEdit.setDisplayFormat("yyyy-MM-dd HH:mm:ss")
        # 设置最小日期
        self.dateEdit.setMinimumDate(
            QDate.currentDate().addDays(-365))

        # 设置最大日期
        self.dateEdit.setMaximumDate(
            QDate.currentDate().addDays(365))
        self.dateEdit.setCalendarPopup( True)

        self.dateEdit.dateChanged.connect(self.onDateChanged)
        self.dateEdit.dateTimeChanged.connect(self.onDateTimeChanged)
        self.dateEdit.timeChanged.connect(self.onTimeChanged)

        self.btn = QPushButton('获得日期和时间')
        self.btn.clicked.connect(self.onButtonClick)

        vlayout.addWidget( self.dateEdit )
        vlayout.addWidget( self.btn )
        self.setLayout(vlayout)

    # 日期发生改变时执行
    def onDateChanged(self , date):
        print(date)

    # 无论是日期还是时间发生改变时都会执行
    def onDateTimeChanged(self , dateTime ):
        print(dateTime)

    # 时间发生改变时执行
    def onTimeChanged(self , time):
        print(time)

    def onButtonClick(self ):

```

```

dateTime = self.dateEdit.dateTime()
#最大日期
maxDate = self.dateEdit.maximumDate()
#最大日期时间
maxDateTime = self.dateEdit.maximumDateTime()
#最大时间
maxTime = self.dateEdit.maximumTime()
#最小日期
minDate = self.dateEdit.minimumDate()
#最小日期时间
minDateTime = self.dateEdit.minimumDateTime()
#最小时间
minTime = self.dateEdit.minimumTime()

print('\n 选择日期时间' )
print('dateTime=%s' % str(dateTime) )
print('maxDate=%s' % str(maxDate) )
print('maxDateTime=%s' % str(maxDateTime) )
print('maxTime=%s' % str(maxTime) )
print('minDate=%s' % str(minDate) )
print('minDateTime=%s' % str(minDateTime) )
print('minTime=%s' % str(minTime) )

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = DateTimeEditDemo()
    demo.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□4-55□□□



```

class:
    def __init__(self):
        self.dateEdit = QDateTimeEdit()
        self.dateEdit.setDisplayFormat("yyyy-MM-dd
HH:mm:ss")

```

## 4.13 菜单和对话框

### 4.13.1 菜单

QMainWindow 类包含 QMenuBar 和 QMenu 类。QMenu 类包含 QAction 类。QMenu 类包含 QAction 类。QMenu 类包含 QAction 类。

PyQt API 中 createPopupMenu() 方法。menuBar() 方法。QMenuBar 类。addMenu() 方法。addAction() 方法。

4-37

方 法	描 述
menuBar()	返回主窗口的 QMenuBar 对象
addMenu()	在菜单栏中添加一个新的 QMenu 对象
addAction()	向 QMenu 小控件中添加一个操作按钮，其中包含文本或图标
setEnabled()	将操作按钮状态设置为启用/禁用
addSeperator()	在菜单中添加一条分隔线
clear()	删除菜单/菜单栏的内容
setShortcut()	将快捷键关联到操作按钮
setText()	设置菜单项的文本
setTitle()	设置 QMenu 小控件的标题
text()	返回与 QAction 对象关联的文本
title()	返回 QMenu 小控件的标题

QAction 和 QMenu 的 triggered 信号

## 4-34 QMenuBar

PyQt5/Chapter04/qt0414\_Qmenu.py PyQt 5 的 QMenuBar、QMenu 和 QAction

```
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
```

```

class MenuDemo(QMainWindow):
    def __init__(self, parent=None):
        super(MenuDemo, self).__init__(parent)
        layout = QHBoxLayout()
        bar = self.menuBar()
        file = bar.addMenu("File")
        file.addAction("New")
        save = QAction("Save", self)
        save.setShortcut("Ctrl+S")
        file.addAction(save)
        edit = file.addMenu("Edit")
        edit.addAction("copy")
        edit.addAction("paste")
        quit = QAction("Quit", self)
        file.addAction(quit)
        file.triggered[QAction].connect(self.processtrigger)
        self.setLayout(layout)
        self.setWindowTitle("menu 例子")

    def processtrigger(self, q):
        print( q.text()+" is triggered" )

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = MenuDemo()
    demo.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□4-56□□□

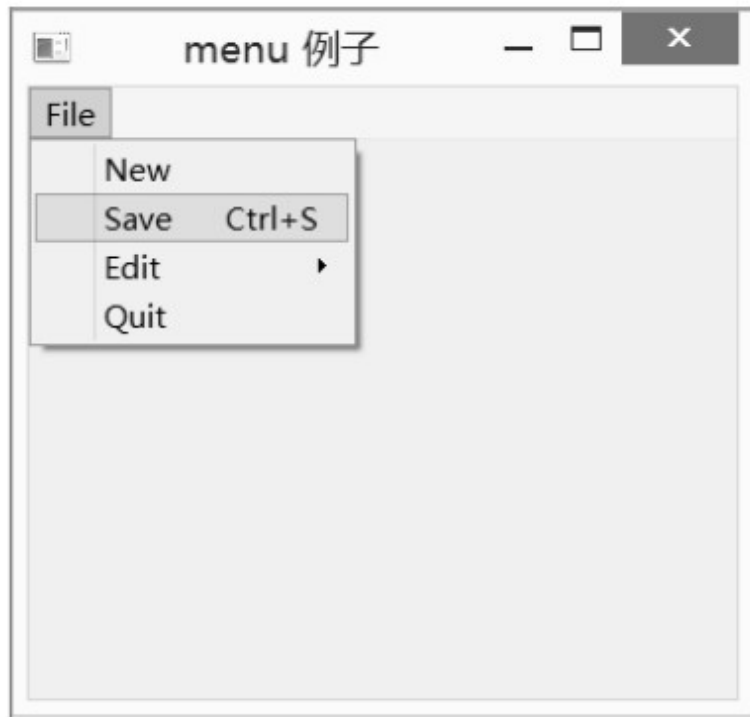


图4-56

代码如下

在 QMainWindow 中添加 QMenuBar 对象

使用 addMenu() 添加“File”菜单

```
bar=self.menuBar()
```

```
file=bar.addMenu("File")
```

在 QMenuBar 对象中添加 QAction 对象

```
file=bar.addMenu("File")
```

```
file.addAction("New")
```

```
save=QAction("Save",self)
```

```
save.setShortcut("Ctrl+S")
```

```
file.addAction(save)
```

在 QAction 对象中添加子菜单

```
edit=file.addAction("Edit")
```

```

edit.addAction("copy")
edit.addAction("paste")

file.triggered.connect(self.processtrigger)

QAction
file.triggered[QAction].connect(self.processtrigger)

```

## 4.13.2 QToolBar

QToolBar 是 Qt 提供的一个工具栏类，它用于在 QMainWindow 窗口中添加工具栏。QToolBar 类继承自 QWidget，并实现了 QAction 接口。

图 4-38

方 法	描 述
addAction()	添加具有文本或图标的工具按钮
addSeparator()	分组显示工具按钮
addWidget()	添加工具栏中按钮以外的控件
addToolBar()	使用 QMainWindow 类的方法添加一个新的工具栏
setMovable()	工具栏变得可移动
setOrientation()	工具栏的方向可以设置为 Qt.Horizontal 或 Qt.vertical

QToolBar 类提供了以下方法：

- addAction()：添加具有文本或图标的工具按钮。
- addSeparator()：在工具栏中添加一个分隔符。
- addWidget()：在工具栏中添加一个 QWidget 子控件。
- addToolBar()：在 QMainWindow 窗口中添加一个新的工具栏。
- setMovable()：设置工具栏是否可移动。
- setOrientation()：设置工具栏的方向，可以是 Qt.Horizontal 或 Qt.Vertical。

### 图 4-35 QToolBar 示例

该示例代码位于 PyQt5/Chapter04/qt0415\_QToolBar.py，展示了如何使用 QToolBar 类。代码中创建了一个 QMainWindow 窗口，并在其中添加了一个工具栏。工具栏中包含两个按钮，分别用于复制和粘贴。当用户点击按钮时，会触发相应的操作。

```

import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class ToolBarDemo( QMainWindow ):

    def __init__(self, parent=None):
        super(ToolBarDemo, self).__init__(parent)
        self.setWindowTitle("toolbar 例子")
        self.resize(300, 200)

        layout = QVBoxLayout()
        tb = self.addToolBar("File")
        new = QAction(QIcon("./images/new.png"), "new", self)
        tb.addAction(new)
        open = QAction(QIcon("./images/open.png"), "open", self)
        tb.addAction(open)
        save = QAction(QIcon("./images/save.png"), "save", self)
        tb.addAction(save)
        tb.actionTriggered[QAction].connect(self.toolbtnpressed)
        self.setLayout(layout)

    def toolbtnpressed(self,a):
        print("pressed tool button is",a.text() )

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = ToolBarDemo()
    demo.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□4-57□□□





图4-57

```

class MainWindow(QMainWindow):
    def __init__(self):
        QMainWindow.__init__(self)
        tb=self.addToolBar("File")
        new=QAction(QIcon("./images/new.png"),"new",self)
        tb.addAction(new)
        open=QAction(QIcon("./images/open.png"),"open",self)
        tb.addAction(open)
        save=QAction(QIcon("./images/save.png"),"save",self)
        tb.addAction(save)
        tb.actionTriggered.connect(self.toolbtnpressed)
        tb.actionTriggered[QAction].connect(self.toolbtnpressed)

```

d)

### [4.13.3 QStatusBar](#)

MainWindow 添加状态栏，并添加一个 QStatusBar 对象。

```
class MainWindow(QMainWindow):
    def __init__(self):
        self.statusBar = QStatusBar()
        self.setStatusBar(self.statusBar)
```

QStatusBar 添加小控件 4-39

图 4-39

方 法	描 述
addWidget()	在状态栏中添加给定的窗口小控件对象
addPermanentWidget()	在状态栏中永久添加给定的窗口小控件对象

图 4-36

方 法	描 述
showMessage()	在状态栏中显示一条临时信息指定时间间隔
clearMessage()	删除正在显示的临时信息
removeWidget()	从状态栏中删除指定的小控件

## 4-36 QStatusBar

在 PyQt5/Chapter04/qt04\_QStatusBar.py 文件中添加以下代码。

```

import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class StatusDemo(QMainWindow):
    def __init__(self, parent=None):
        super(StatusDemo, self).__init__(parent)
        bar = self.menuBar()
        file = bar.addMenu("File")
        file.addAction("show")
        file.triggered[QAction].connect(self.processTrigger)
        self.setCentralWidget(QTextEdit())
        self.statusBar = QStatusBar()
        self.setWindowTitle("QStatusBar 例子")
        self.setStatusBar(self.statusBar)

    def processTrigger(self, q):
        if (q.text()=="show"):
            self.statusBar.showMessage(q.text()+" 菜单选项被点击了", 5000)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = StatusDemo()
    demo.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□4-58□□□



图4-58

```

class MainWindow(QMainWindow)
{
public:
    MainWindow(QWidget *parent) : QMainWindow(parent)
    {
        file.triggered[QAction].connect(self.processTrigger)
        statusBar->showMessage("show 菜单选项被点击了", 5000)
    }
private:
    void processTrigger()
    {
        if (q.text()=="show"):
            self.statusBar.showMessage(q.text()+" 菜单选项被点击了", 5000)
    }
}

```

## 4.14 QPrinter

```
QPainter::QPainter(QPaintDevice *device) : QAbstractPainter(device, new QPainter()) {}  
QPainter::~QPainter() {}  
QPainter & QPainter::operator=(const QPainter &painter) { return *this; }  
QPainter * QPainter::clone() const { return new QPainter(*this); }  
QPainter * QPainter::clone(QPaintDevice *device) const { return new QPainter(device, *this); }  
QPainter * QPainter::clone(QImage *image) const { return new QPainter(image, *this); }  
QPainter * QPainter::clone(QPrinter *printer) const { return new QPainter(printer, *this); }
```

## 4-37 QPrinter

```
PyQt5/Chapter04/qt04Painter.py QPainter
PyQt5/Chapter04/qt04Painter.py
```

```
from PyQt5.QtGui import QImage , QIcon, QPixmap
from PyQt5.QtWidgets import QApplication , QMainWindow, QLabel,
QSizePolicy , QAction
from PyQt5.QtPrintSupport import QPrinter, QPrintDialog
import sys
```

```

class MainWindow(QMainWindow):
    def __init__(self, parent=None):
        super(MainWindow, self).__init__(parent)
        self.setWindowTitle(self.tr("打印图片"))
        self.imageLabel=QLabel()
        self.imageLabel.setSizePolicy(
            QSizePolicy.Ignored, QSizePolicy.Ignored)
        self.setCentralWidget(self.imageLabel)
        self.image=QImage()
        self.createActions()
        self.createMenus()
        self.createToolBars()

        if self.image.load("./images/screen.png"):
            self.imageLabel.setPixmap(QPixmap.fromImage(self.image))
            self.resize(self.image.width(), self.image.height())

    def createActions(self):
        self.PrintAction=QAction(
            QIcon("./images/printer.png"),
            self.tr("打印"),
            self )
        self.PrintAction.setShortcut("Ctrl+P")
        self.PrintAction.setStatusTip(self.tr("打印"))
        self.PrintAction.triggered.connect(self.slotPrint)

    def createMenus(self):
        PrintMenu=self.menuBar().addMenu(self.tr("打印"))
        PrintMenu.addAction(self.PrintAction)

    def createToolBars(self):
        fileToolBar=self.addToolBar("Print")
        fileToolBar.addAction(self.PrintAction)

    def slotPrint(self):
        printer=QPrinter()
        printDialog=QPrintDialog(printer, self)
        if printDialog.exec_():
            painter=QPainter(printer)
            rect=painter.viewport()
            size=self.image.size()
            size.scale(rect.size(), Qt.KeepAspectRatio)

```

```

        painter.setViewport(rect.x(),rect.y(),size.width(),
size.height())
        painter.setWindow(self.image.rect())
        painter.drawImage(0,0,self.image)

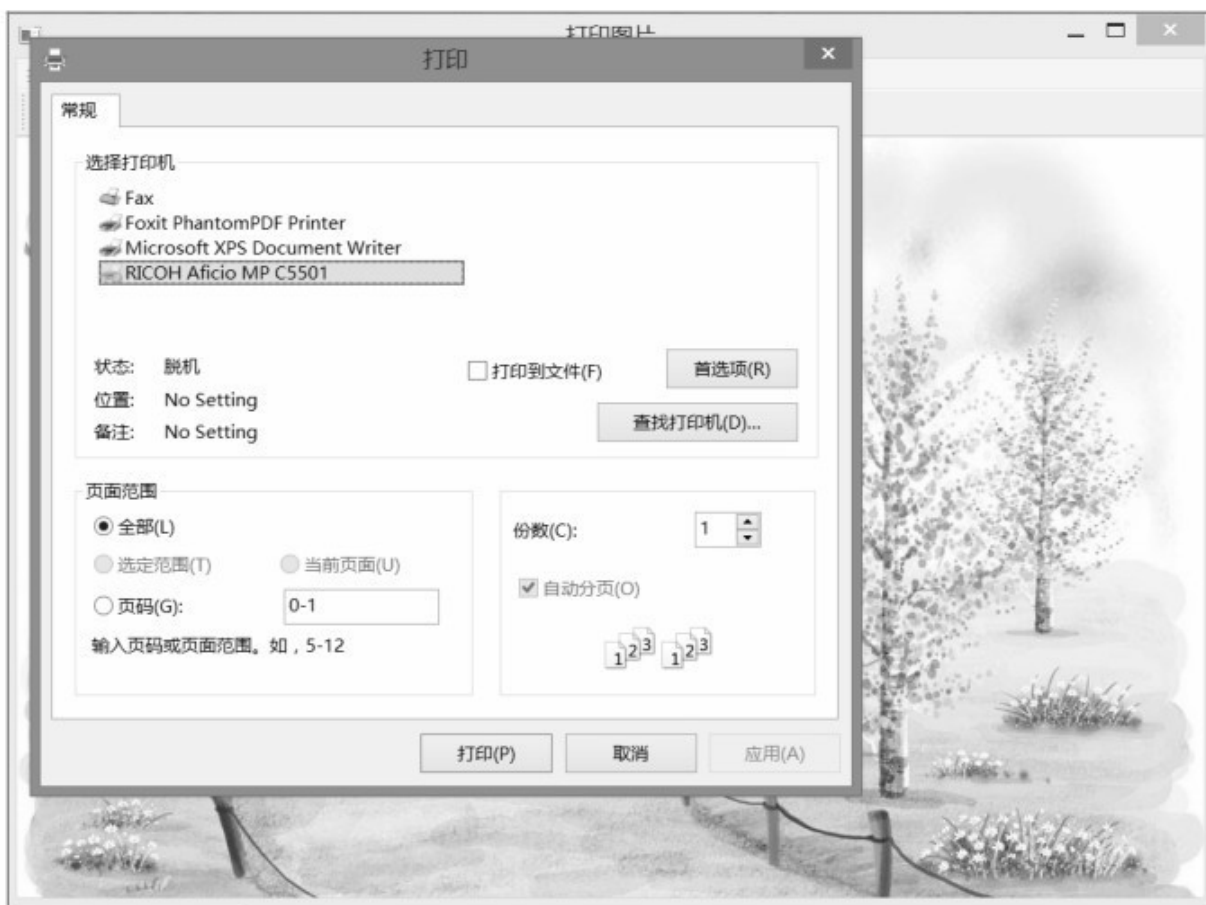
if __name__ == "__main__":
    app=QApplication(sys.argv)
    main=MainWindow()
    main.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□4-59□□4-60□□□



□4-59





## 5 PyQt 5

## 5.1 ☐☐☐☐

PyQt

### 5.1.1 QTableView

[illegible]

# QTableView5-1



## PyQt5/Chapter05/qt05\_tblViewModel.py

```
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
from PyQt5.QtCore import *
import sys

class Table(QWidget):

    def __init__(self, arg=None):
        super(Table, self).__init__(arg)
        self.setWindowTitle("QTableView 表格视图控件的例子")
        self.resize(500,300);
        self.model=QStandardItemModel(4,4);
        self.model.setHorizontalHeaderLabels(['标题 1','标题 2','标题
3','标题 4'])

        for row in range(4):
            for column in range(4):
                item = QStandardItem("row %s, column %s"%(row,column))

                self.model.setItem(row, column, item)

        self.tableView=QTableView()
        self.tableView.setModel(self.model)

        dlgLayout=QVBoxLayout();
        dlgLayout.addWidget(self.tableView)
        self.setLayout(dlgLayout)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    table = Table()
    table.show()
    sys.exit(app.exec_())
```

5-2



图5-2

图 5-2 表格视图控件的例子

1. 设置表格视图的列宽

```
self.tableView.horizontalHeader().setStretchLastSection(
True)
```

```
self.tableView.horizontalHeader().setSectionResizeMod
e(QHeaderView.Stretch)
```

2. 添加数据

```
self.model.appendRow([
    QTableWidgetItem("row %s,column %s"%(1,1)),
    QTableWidgetItem("row %s,column %s"%(1,1)),
    QTableWidgetItem("row %s,column %s"%(1,1)),
    QTableWidgetItem("row %s,column %s"%(1,1)),
])
```

3. 设置表格视图的列宽

```

def delete():
    # 删除一行
    indexs=self.tableView.selectionModel().selection().indexes()
    if len(indexs)>0:
        # 删除第一行
        index=indexs[0]
        self.model.removeRows(index.row(),1)
    # 更新视图
    index=self.tableView.currentIndex()
    print(index.row())
    self.model.removeRow(index.row())
    # 删除一行后，表格行数减1，所以需要更新表格的行数
    self.tableView.model().rowCount()

```

### 5.1.2 QListView

```
QListView □□□□□□□□□□ QListWidget □ QListView □□□□□  
□ Model □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□  
QListWidget □□□□□□□□ QListView □□□□□□□□□□□□□□□□□□□□  
□ QListWidgetItem □□□□□□ addItem() □□□□□□□□□□□□□□ Item □□  
QListView □□□□□□□□□□ 5-2 □□□
```

□5-2

方 法	描 述
setModel()	用来设置 View 所关联的 Model, 可以使用 Python 原生的 list 作为数据源 Model
selectedItem()	选中 Model 中的条目
isSelected()	判断 Model 中的某条目是否被选中

# QListView 5-3

5-3

信 号	含 义
clicked	当单击某项时，信号被发射
doubleClicked	当双击某项时，信号被发射

## 5-2 QListView

PyQt5/Chapter05/qt05\_listView.py PyQt 5  
QListView

```

from PyQt5.QtWidgets import QApplication, QWidget , QVBoxLayout ,
QListView, QMessageBox
from PyQt5.QtCore import QStringListModel
import sys

class ListViewDemo(QWidget):
    def __init__(self, parent=None):
        super(ListViewDemo, self).__init__(parent)
        self.setWindowTitle("QListView 例子")
        self.resize(300, 270)
        layout = QVBoxLayout()

        listView = QListView()
        slm = QStringListModel();
        self.qList = ['Item 1','Item 2','Item 3','Item 4' ]
        slm.setStringList(self.qList)
        listView.setModel(slm )
        listView.clicked.connect(self.clicked)
        layout.addWidget( listView )
        self.setLayout(layout)

    def clicked(self, qModelIndex):
        QMessageBox.information(self, "ListWidget", "你选择了: "+
self.qList[qModelIndex.row()])

if __name__ == "__main__":
    app = QApplication(sys.argv)
    win = ListViewDemo()
    win.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□5-3□□□

□□□□□□□□□□QListView□□□Model□□□□□□□□□□□□□□□□□□□□

□□□

□QListView□□□clicked□□□□□□□□□clicked()□□□□□□□□□

listView.clicked.connect(self.clicked)



图5-3

### 5.1.3 QListViewWidget

QListWidget 是 QListView 的子类，它提供了更丰富的列表功能。QListWidget 是 QListView 的子类，它提供了更丰富的列表功能。QListWidget 是 QListView 的子类，它提供了更丰富的列表功能。

图5-4

方 法	描 述
addItem()	在列表中添加 QListWidgetItem 对象或字符串
addItems()	添加列表中的每个条目
insertItem()	在指定的索引处插入条目
clear()	删除列表的内容
setCurrentItem()	设置当前所选条目
sortItems()	按升序重新排列条目

QListWidget 是 QListView 的子类，它提供了更丰富的列表功能。

图5-5



信 号	含 义
currentItemChanged	当列表中的条目发生改变时发射此信号
itemClicked	当点击列表中的条目时发射此信号

### 5-3 QListWidget

PyQt5/Chapter05/qt05\_QListWidget.py PyQt 5  
 QListWidget

```
import sys
from PyQt5.QtCore import *

from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class ListWidget(QListWidget):
    def clicked(self, item):
        QMessageBox.information(self, "ListWidget", "你选择了: "
                                + item.text())

if __name__ == '__main__':
    app = QApplication(sys.argv)
    listWidget = ListWidget()
    listWidget.resize(300, 120)
    listWidget.addItem("Item 1");
    listWidget.addItem("Item 2");
    listWidget.addItem("Item 3");
    listWidget.addItem("Item 4");
    listWidget.setWindowTitle('QListwidget 例子')
    listWidget.itemClicked.connect(listWidget.clicked)
    listWidget.show()
    sys.exit(app.exec_())
```

5-4



方 法	描 述
setRowCount(int row)	设置 QTableWidgetItem 表格控件的行数
setColumnCount(int col)	设置 QTableWidgetItem 表格控件的列数
setHorizontalHeaderLabels()	设置 QTableWidgetItem 表格控件的水平标签
setVerticalHeaderLabels()	设置 QTableWidgetItem 表格控件的垂直标签
setItem(int, int, QTableWidgetItem)	在 QTableWidgetItem 表格控件的每个选项的单元空间里添加控件
horizontalHeader()	获得 QTableWidgetItem 表格控件的表格头，以便执行隐藏
rowCount()	获得 QTableWidgetItem 表格控件的行数
columnCount()	获得 QTableWidgetItem 表格控件的列数
setEditTriggers(EditTriggers triggers)	设置表格是否可编辑。设置编辑规则的枚举值
setSelectionBehavior	设置表格的选择行为
setTextAlignment()	设置单元格内文字的对齐方式
setSpan(int row, int column, int rowSpanCount, int columnSpanCount)	合并单元格，要改变单元格的第 row 行第 column 列，要合并 rowSpanCount 行数和 columnSpanCount 列数。 <ul style="list-style-type: none"> <li>• row: 要改变的单元格行数</li> <li>• column: 要改变的单元格列数</li> <li>• rowSpanCount: 需要合并的行数</li> <li>• columnSpanCount: 需要合并的列数</li> </ul>
setShowGrid()	在默认情况下，表格的显示是有网格线的。 <ul style="list-style-type: none"> <li>• True: 显示网格线</li> <li>• False: 不显示网格线</li> </ul>
setColumnWidth(int column, int width)	设置单元格行的宽度
setRowHeight(int row, int height)	设置单元格列的高度

5-7

5-7

选 项	值	描 述
QAbstractItemView.NoEditTriggers0No	0	不能对表格内容进行修改
QAbstractItemView.CurrentChanged1Editing	1	任何时候都能对单元格进行修改
QAbstractItemView.DoubleClicked2Editing	2	双击单元格
QAbstractItemView.SelectedClicked4Editing	4	单击已选中的内容
QAbstractItemView.EditKeyPressed8Editing	8	当修改键被按下时修改单元格
QAbstractItemView.AnyKeyPressed16Editing	16	按任意键修改单元格
QAbstractItemView.AllEditTriggers31Editing	31	包括以上所有条件

5-8

5-8



```

import sys
from PyQt5.QtWidgets import (QWidget, QTableWidgetItem, QHBoxLayout,
QApplication, QTableWidgetItem )

class Table(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

```

```

    def initUI(self):
        self.setWindowTitle("QTableWidget 例子")
        self.resize(400,300);
        conLayout = QHBoxLayout()
        tableWidget=QTableWidget()
        tableWidget.setRowCount(4)
        tableWidget.setColumnCount(3)
        conLayout.addWidget(tableWidget )
        tableWidget.setHorizontalHeaderLabels(['姓名','性别','体重
(kg)'])

        newItem = QTableWidgetItem("张三")
        tableWidget.setItem(0, 0, newItem)

        newItem = QTableWidgetItem("男")
        tableWidget.setItem(0, 1, newItem)

        newItem = QTableWidgetItem("160")
        tableWidget.setItem(0, 2, newItem)

        self.setLayout(conLayout)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    example = Table()
    example.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□5-5□□□



图5-5

```

import sys
from PyQt5.QtWidgets import QApplication, QMainWindow, QTableWidget, QTableWidgetItem

class Example(QMainWindow):
    def __init__(self):
        super().__init__()
        self.table = QTableWidget(4, 3)
        self.table.setHorizontalHeaderLabels(['姓名', '性别', '体重(kg)'])
        self.setWindowTitle("QTableWidget 例子")
        self.setCentralWidget(self.table)
        newItem = QTableWidgetItem("张三")
        self.table.setItem(0, 0, newItem)
        self.table.setItem(0, 1, QTableWidgetItem("男"))
        self.table.setItem(0, 2, QTableWidgetItem("160"))
        self.show()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    ex.show()
    sys.exit(app.exec_())

```

```

##### PyQt5/Chapter05/qt05_tblHeader.py#####
#####
#####QTableWidget#####4*3#####
    tableWidget=QTableWidget()
    tableWidget.setRowCount(4)
    tableWidget.setColumnCount(3)
    #####
    tableWidget.setHorizontalHeaderLabels(['姓名','性别','体重(kg)'])
    tableWidget.setVerticalHeaderLabels(['行 1','行 2','行 3','行 4'])
    #####
    #####
    #####5-6#####

```



	姓名	性别	体重(kg)
行1	张三	男	160
行2			
行3			
行4			

□2□□□□□□□□□□

```

    000040300000000000“”“”kg”0000000000000000
    0000000QTableWidgetItem 00horizontalHeager()000000000000
    00000000000000000000

```

```
tableWidget=QTableWidget()
tableWidget.setRowCount(4)
tableWidget.setColumnCount(3)
tableWidget.setHorizontalHeaderLabels([' ',' ',' '
(kg)'])
tableWidget.horizontalHeader().
    setSectionResizeMode(QHeaderView.Stretch)
5-7
```





❸ ❶ ❷ ❸ ❹ ❺ ❻ ❼ ❽ ❾ ❿ ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ⓵ ⓶ ⓷ ⓸ ⓹ ⓺ ⓻ ⓼ ⓽ ⓾ ⓿  
 ❹ ❶ ❷ ❸ ❹ ❺ ❻ ❼ ❽ ❾ ❿ ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ⓵ ⓶ ⓷ ⓸ ⓹ ⓺ ⓻ ⓼ ⓽ ⓾ ⓿  
 ❶ ❷ ❸ ❹ ❺ ❻ ❼ ❽ ❾ ❿ ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ⓵ ⓶ ⓷ ⓸ ⓹ ⓺ ⓻ ⓼ ⓽ ⓾ ⓿  
 tableWidget.setEditTriggers(QAbstractItemView.NoEditTriggers)  
 ❹ ❶ ❷ ❸ ❹ ❺ ❻ ❼ ❽ ❾ ❿ ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ⓵ ⓶ ⓷ ⓸ ⓹ ⓺ ⓻ ⓼ ⓽ ⓾ ⓿  
 ❶ ❷ ❸ ❹ ❺ ❻ ❼ ❽ ❾ ❿ ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ⓵ ⓶ ⓷ ⓸ ⓹ ⓺ ⓻ ⓼ ⓽ ⓾ ⓿  
 tableWidget.setSelectionBehavior(QAbstractItemView.SelectRows)  
 ❶ ❷ ❸ ❹ ❺ ❻ ❼ ❽ ❾ ❿ ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ⓵ ⓶ ⓷ ⓸ ⓹ ⓺ ⓻ ⓼ ⓽ ⓾ ⓿  
 ❶ ❷ ❸ ❹ ❺ ❻ ❼ ❽ ❾ ❿ ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ⓵ ⓶ ⓷ ⓸ ⓹ ⓺ ⓻ ⓼ ⓽ ⓾ ⓿  
 QTableWidgetItem.resizeColumnsToContents()  
 QTableWidgetItem.resizeRowsToContents()  
 ❶ ❷ ❸ ❹ ❺ ❻ ❼ ❽ ❾ ❿ ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱ ⓲ ⓳ ⓴ ⓵ ⓶ ⓷ ⓸ ⓹ ⓺ ⓻ ⓼ ⓽ ⓾ ⓿

	姓名	性别	体重(kg)
1	张三	男	160
2			
3			
4			



图5-9

图6展示了如何隐藏

表头。通过调用以下代码：

```
tableWidget.verticalHeader().setVisible(False)
```

可以隐藏垂直表头。

```
tableWidget.horizontalHeader().setVisible(False)
```

可以隐藏水平表头。图5-10展示了

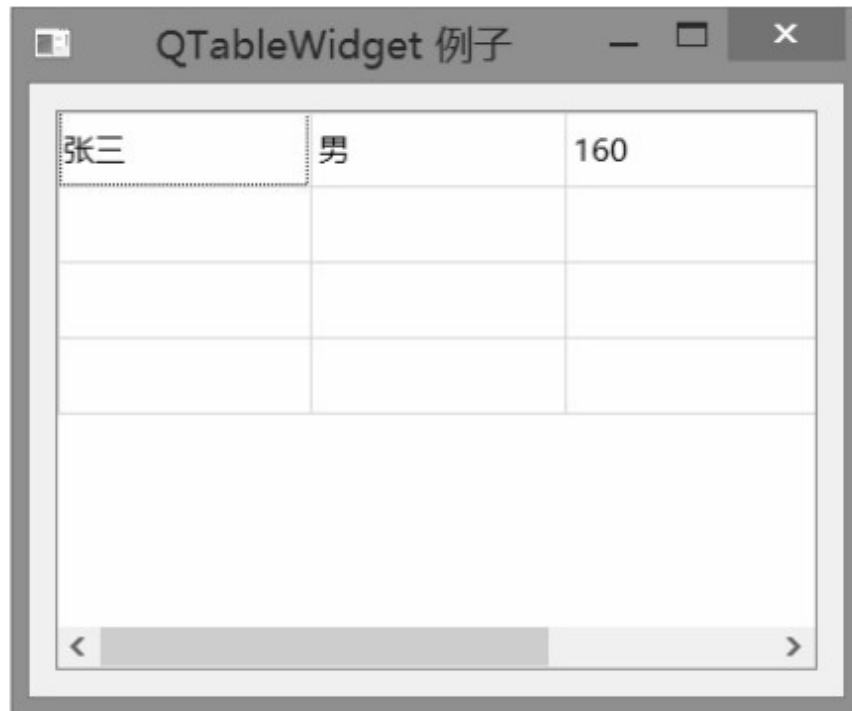


图5-10

例7 添加自定义控件

QTableWidget 添加自定义控件

TableWidget.setItem() PyQt

PyQt5/Chapter05/qt05\_tblCmb.py

3px

```
comboBox=QComboBox()
```

```
comboBox.addItem("")
```

```
comboBox.addItem("")
```

```
comboBox.setStyleSheet("QComboBox{margin:3px};")
```

```
tableWidget.setCellWidget(0,1,comboBox)
```

```
searchBtn=QPushButton("搜索")
```

```
searchBtn.setDown( True )
```

```
searchBtn.setStyleSheet("QPushButton{margin:3px};")
```

```
tableWidget.setCellWidget(0,2,searchBtn)
```

图5-11



图5-11

图8

tableWidget

10

#

item=self.tableWidget.findItems(text,QtCore.Qt.MatchExactly

#

row=item[0].row()

#

self.tableWidget.verticalScrollBar().setSliderPosition(row)

PyQt5/Chapter05/qt05\_tbSellItem.py

tableWidget

```
import sys
from PyQt5.QtWidgets import *
from PyQt5 import QtCore
from PyQt5.QtGui import QColor , QBrush

class Table(QWidget):
    def __init__(self):
```

```

super().__init__()
self.initUI()

def initUI(self):
    self.setWindowTitle("QWidget 例子")
    self.resize(600,800);
    conLayout = QHBoxLayout()
    tableWidget = QWidget()
    tableWidget.setRowCount(30)
    tableWidget.setColumnCount(4)
    conLayout.addWidget(tableWidget )

    for i in range(30):
        for j in range(4):
            itemContent = '(%d,%d)' % (i,j)
            tableWidget.setItem(i,j, QTableWidgetItem(itemContent))
    self.setLayout(conLayout)

    #遍历表格查找对应项
    text = "(10,1)"
    items = tableWidget.findItems(text, QtCore.Qt.MatchExactly)
    item = items[0]
    # 选中单元格
    item.setSelected( True)
    # 设置单元格的背景颜色为红色
    item.setForeground(QBrush(QColor(255, 0, 0)))

    row = item.row()
    #通过鼠标滚轮定位，快速定位到第 11 行
    tableWidget.verticalScrollBar().setSliderPosition(row)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    example = Table()
    example.show()
    sys.exit(app.exec_())

```

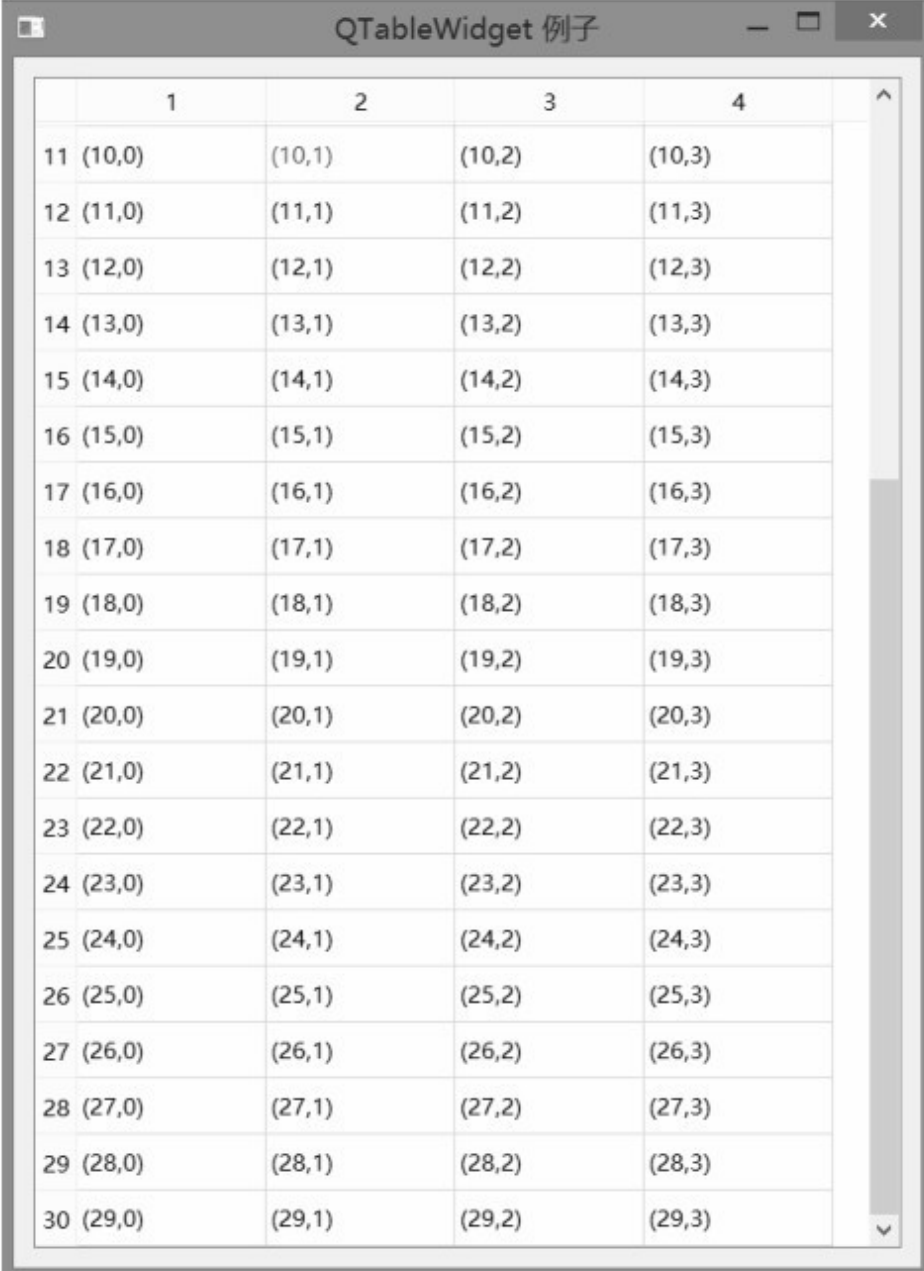
□□□□□□□□□□5-12□□□

## 2. 例子

1. 例子

PyQt5/Chapter05/qt05\_tblItemColor.py

PyQt5/Chapter05/qt05\_tblItemColor.py



	1	2	3	4
11	(10,0)	(10,1)	(10,2)	(10,3)
12	(11,0)	(11,1)	(11,2)	(11,3)
13	(12,0)	(12,1)	(12,2)	(12,3)
14	(13,0)	(13,1)	(13,2)	(13,3)
15	(14,0)	(14,1)	(14,2)	(14,3)
16	(15,0)	(15,1)	(15,2)	(15,3)
17	(16,0)	(16,1)	(16,2)	(16,3)
18	(17,0)	(17,1)	(17,2)	(17,3)
19	(18,0)	(18,1)	(18,2)	(18,3)
20	(19,0)	(19,1)	(19,2)	(19,3)
21	(20,0)	(20,1)	(20,2)	(20,3)
22	(21,0)	(21,1)	(21,2)	(21,3)
23	(22,0)	(22,1)	(22,2)	(22,3)
24	(23,0)	(23,1)	(23,2)	(23,3)
25	(24,0)	(24,1)	(24,2)	(24,3)
26	(25,0)	(25,1)	(25,2)	(25,3)
27	(26,0)	(26,1)	(26,2)	(26,3)
28	(27,0)	(27,1)	(27,2)	(27,3)
29	(28,0)	(28,1)	(28,2)	(28,3)
30	(29,0)	(29,1)	(29,2)	(29,3)

```

newItem=QTableWidgetItem("")
newItem.setForeground(QBrush(QColor(255,0,0)))
tableWidget.setItem(0,0,newItem)
newItem=QTableWidgetItem("")
newItem.setForeground(QBrush(QColor(255,0,0)))
tableWidget.setItem(0,1,newItem)
newItem=QTableWidgetItem("160")
newItem.setForeground(QBrush(QColor(255,0,0)))
tableWidget.setItem(0,2,newItem)

```

图5-13



图5-13

2. 设置字体

文件路径PyQt5/Chapter05/qt05\_tblItemFont.py

```

newItem=QTableWidgetItem("")
newItem.setFont( QFont( "Times",12,QFont.Black ) )
tableWidget.setItem(0,0,newItem)
newItem=QTableWidgetItem("")

```



```

newItem.setFont( QFont( "Times",12,QFont.Black ) )
tableWidget.setItem(0,1,newItem)
newItem=QTableWidgetItem("160")
newItem.setFont( QFont( "Times",12,QFont.Black ) )
tableWidget.setItem(0,2,newItem)

```

□□□□□5-14□□□



	姓名	性别	体重(kg)
1	张三	男	160
2			
3			
4			

□5-14

□3□□□□□□□□□□

□□ Qt □□□□□<https://doc.qt.io/qt-5/qt.html#details>□□□□□□

□□Qt.DescendingOrder□□□□□□□□□□□□ Qt.AscendingOrder□□

□□□□□□□□□□□□□□□□□□PyQt5.QtCore□□□□Qt□□

```
from PyQt5.QtCore import Qt
```

□□□□□PyQt5/Chapter05/qt05\_tblItemOrder.py□□□□□□□□□□

□□□□□□□□□□□□□□□□□□

```
# Qt.DescendingOrder □□
```

```
# Qt.AscendingOrder □□
```

```
tableWidget.sortItems(2,QtCore.Qt.DescendingOrder )
```





图5-16

图5-16 表格的跨行

图5-16 表格的跨行

```
tableWidget.setSpan(0,0,3,1)
```

图5-16 表格的跨行

```
tableWidget=QTableWidget()
```

```
tableWidget.setRowCount(4)
```

```
tableWidget.setColumnCount(3)
```

```
tableWidget.setHorizontalHeaderLabels(['姓名','性别','体重(kg)'])
```

```
tableWidget.setSpan(0,0,3,1)
```

```
newItem=QTableWidgetItem("张三")
```

```
tableWidget.setItem(0,0,newItem)
```

```
newItem=QTableWidgetItem("男")
```

```
tableWidget.setItem(0,1,newItem)
```

```
newItem=QTableWidgetItem("160")
```

```
tableWidget.setItem(0,2,newItem)
```

图5-17



图5-17

图6

PyQt5/Chapter05/ qt05\_tblRow.py

150

```
#
```

```
tableWidget.setColumnWidth(0,150)
```

```
#
```

```
tableWidget.setRowHeight(0,120)
```

图5-18



图5-18

图7展示了如何隐藏

QTableWidget 的 `setShowGrid()` 方法在 QTableView 中

可以隐藏表格的网格线，代码如下所示：

```
tableWidget.setShowGrid(False)
```

图5-19展示了



图5-19

通过调用 `tableWidget.verticalHeader().setVisible(False)` 可以隐藏行索引。

图5-20展示了隐藏行索引后的效果。

图5-20



图5-20

例8 添加新行

在已有的表格中添加一行，并在新行中添加一个图片项。

PyQt5/Chapter05/qt05\_tblItemIcon01.py 示例程序

```
newItem=QTableWidgetItem(QIcon("./images/bao1.png"),"背包")
```

```
self.tableWidget.setItem(0,3,newItem )
```

图5-21 示例



图5-21

例9 添加新列

在已有的表格中添加一列，并在新列中添加一个图片项。

PyQt5/Chapter05/qt05\_tblItemIcon02.py 示例程序

```

conLayout = QHBoxLayout()

table= QTableWidgetItem()
table.setColumnCount(3)
table.setRowCount(5)

table.setHorizontalHeaderLabels(['图片 1','图片 2','图片 3'])

table.setEditTriggers( QAbstractItemView.NoEditTriggers)
table.setIconSize(QSize(300,200));

for i in range(3): # 让列宽和图片相同
    table.setColumnWidth(i , 300)
for i in range(5): # 让行高和图片相同
    table.setRowHeight(i , 200)

for k in range(15): # 模拟产生 15 条记录
    i = k/3
    j = k%3
    item = QTableWidgetItem()
    item.setFlags(Qt.ItemIsEnabled) #用户点击表格时，图片被选中
    icon = QIcon(r'..\images\bao%d.png' % k )
    item.setIcon(QIcon(icon) )

    print('e/icons/%d.png i=%d j=%d' %( k , i , j ) )
    table.setItem(i,j,item)

```

```

conLayout.addWidget( table)
self.setLayout(conLayout)

```

□□□□□5-22□□□



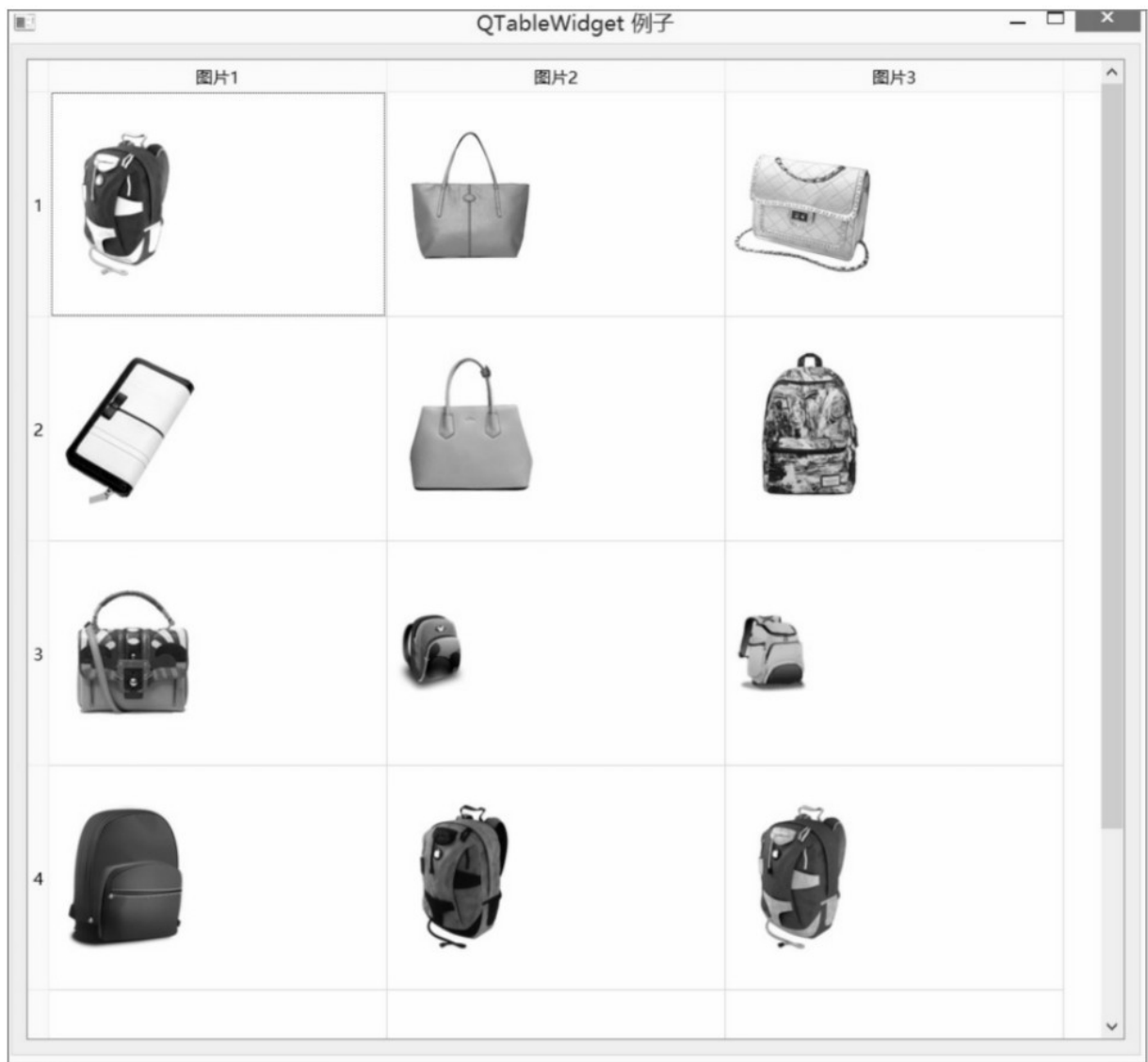


图5-22

```

10
def itemClicked ( QTableWidgetItem *)
    itemClicked.connect( self.getItem()
    tableWidget.itemClicked.connect( self.handleItemClick
    )
def getItem(self,item):
    print('you selected=' + item.text())

```

### 3. 실행결과

실행결과 PyQt5/Chapter05/qt05\_tblMenu.py 실행결과 화면

```
import sys

from PyQt5.QtWidgets import ( QMenu, QPushButton, QWidget, QTableWidget,
QHBoxLayout, QApplication, QDesktopWidget, QTableWidgetItem, QHeaderView)

from PyQt5.QtCore import pyqtSignal, QObject, Qt, pyqtSlot


class Table( QWidget ):

    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        self.setWindowTitle("QTableWidget demo")
        self.resize(500,300);
        conLayout = QHBoxLayout()
        self.tableWidget= QTableWidget()
        self.tableWidget.setRowCount(5)
        self.tableWidget.setColumnCount(3)
        conLayout.addWidget(self.tableWidget )

        self.tableWidget.setHorizontalHeaderLabels(['姓名','性别','体重' ])
        self.tableWidget.horizontalHeader().
            setSectionResizeMode(QHeaderView.Stretch)

        newItem = QTableWidgetItem("张三")
        self.tableWidget.setItem(0, 0, newItem)

        newItem = QTableWidgetItem("男")
        self.tableWidget.setItem(0, 1, newItem)

        newItem = QTableWidgetItem("160")
        self.tableWidget.setItem(0, 2, newItem)
        # 表格中第二行记录
        newItem = QTableWidgetItem("李四")
        self.tableWidget.setItem(1, 0, newItem)

        newItem = QTableWidgetItem("女")
        self.tableWidget.setItem(1, 1, newItem)

        newItem = QTableWidgetItem("170")
        self.tableWidget.setItem(1, 2, newItem)
        # 允许右键产生菜单
```

```

        self.tableWidget.setContextMenuPolicy(Qt.CustomContextMenu)
        # 将右键菜单绑定到槽函数 generateMenu
        self.tableWidget.customContextMenuRequested.
            connect(self.generateMenu)
        self.setLayout(conLayout)

    def generateMenu(self, pos):
        row_num = -1
        for i in self.tableWidget.selectionModel().selection().indexes():
            row_num = i.row()

        # 表格中只有两条有效数据，所以只在前两行支持右键弹出菜单
        if row_num < 2 :
            menu = QMenu()
            item1 = menu.addAction(u"选项一")
            item2 = menu.addAction(u"选项二")
            item3 = menu.addAction(u"选项三" )
            action = menu.exec_(self.tableWidget.mapToGlobal(pos))
            if action == item1:
                print( '您选了选项一，当前行文字内容是: ',self.tableWidget.
item(row_num,0).text(),self.tableWidget.item(row_num,1).text() ,self.tab
leWidget.item(row_num,2).text())

                elif action == item2:
                    print( '您选了选项二，当前行文字内容是: ',self.tableWidget.
item(row_num,0).text(),self.tableWidget.item(row_num,1).text() ,self.tab
leWidget.item(row_num,2).text() )

                    elif action == item3:
                        print( '您选了选项三，当前行文字内容是: ', self.tableWidget.
item(row_num,0).text(),self.tableWidget.item(row_num,1).text() ,self.tab
leWidget.item(row_num,2).text() )
                    else:
                        return

        if __name__ == '__main__':
            app = QApplication(sys.argv)
            example = Table()
            example.show()
            sys.exit(app.exec_())

```

5-23



	姓名	性别	体重
1	张三	男	160
2	李四	女	170
3			
4			
5			

5-23

“”  
160

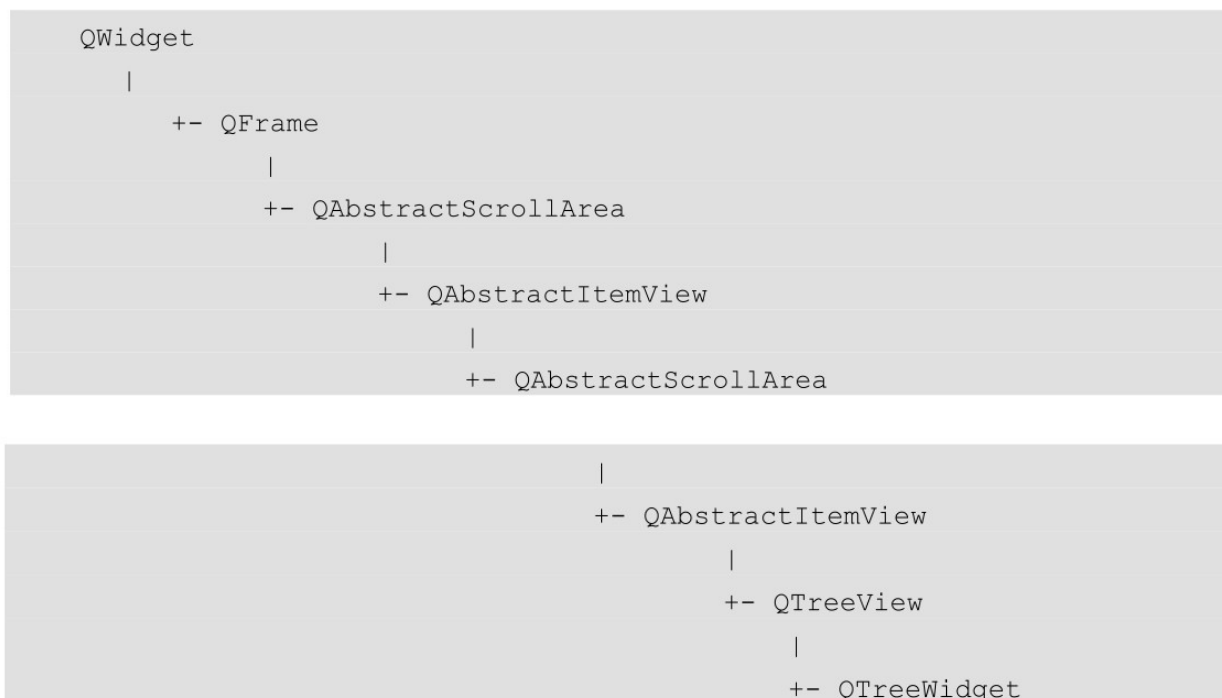
5.1.5 QTreeView

QTreeWidget 5-24



图5-24

QTreeWidget的类层次结构



QTreeWidget的类层次结构图5-11

图5-11

方 法	描 述
setColumnWidth(int column, int width)	将指定列的宽度设置为给定的值 Column, 指定的列 Width, 指定列的宽度
insertTopLevelItems()	在视图的顶层索引中插入项目列表
expandAll()	展开所有的树形节点
invisibleRootItem()	返回树形控件中不可见的根选项 (Root Item)
selectedItems()	返回所有选定的非隐藏项目的列表

## QTreeWidgetItem 5-12

### 5-12

方 法	描 述
addChild()	将子项追加到子列表中
setText()	设置显示的节点文本
Text()	返回显示的节点文本
setCheckState(column, state)	设置指定列的选中状态: Qt.Checked, 节点选中 Qt.Unchecked, 节点未选中
setIcon(column, icon)	在指定的列中显示图标

### 1. 1.1

1.1.1 QTreeWidgetItem 1.1.1  
QTreeWidgetItem 1.1.1  
1.1.1 PyQt5/Chapter05/qt05\_treewidget01.py 1.1.1  
1.1.1

```

self.tree=QTreeWidgetItem()
# 
self.tree.setColumnCount(2)
# 
self.tree.setHeaderLabels(['Key','Value'])
# 
root=QTreeWidgetItem(self.tree)
root.setText(0,'root')

```

```

root.setIcon(0,QIcon("./images/root.png"))
# 初始化根节点
self.tree.setColumnWidth(0,160)
# 初始化子节点1
child1=QTreeWidgetItem(root)
child1.setText(0,'child1')
child1.setText(1,'ios')
child1.setIcon(0,QIcon("./images/IOS.png"))
# 初始化子节点2
child2=QTreeWidgetItem(root)
child2.setText(0,'child2')
child2.setText(1,'')
child2.setIcon(0,QIcon("./images/android.png"))
# 初始化子节点3
child3=QTreeWidgetItem(child2)
child3.setText(0,'child3')
child3.setText(1,'android')
child3.setIcon(0,QIcon("./images/music.png"))
self.tree.addTopLevelItem(root)
# 展开树
self.tree.expandAll()
# 插入顶层项
QTreeWidgetItem.insertTopLevelItems()
# 初始化树
self.tree=QTreeWidgetItem()
# 初始化
self.tree.setColumnCount(2)
# 初始化

```



```

self.tree.setHeaderLabels(['Key','Value'])
# 初始化
root=QTreeWidgetItem()
root.setText(0,'root')
rootList=[]
rootList.append(root)
# 添加子项1
child1=QTreeWidgetItem()
child1.setText(0,'child1')
child1.setText(1,'ios')
root.addChild(child1)
self.tree.insertTopLevelItems(0,rootList)
1. 初始化
2. 初始化QTreeWidgetItem并设置初始状态
3. 初始化
    child1=QTreeWidgetItem(root)
    child1.setText(0,'child1')
    child1.setText(1,'ios')
    child1.setIcon(0,QIcon("./images/IOS.png"))
    child1.setCheckState(0,Qt.Checked)
4. 5-25

```



05-25

2. 创建树形结构

使用QBrush设置背景颜色

```

root=QTreeWidgetItem(self.tree)
root.setText(0,'root')
root.setIcon(0,QIcon("./images/root.png"))
brush_red=QBrush(Qt.red)
root.setBackground(0,brush_red)
brush_green=QBrush(Qt.green)
root.setBackground(1,brush_green)

```

2. 创建树形结构

在PyQt5/Chapter05/qt05\_treewidget02.py中添加以下代码

在main函数中添加以下代码

```
from PyQt5.QtWidgets import *
import sys

class TreeWidgetDemo(QMainWindow):
    def __init__(self, parent=None):
        super(TreeWidgetDemo, self).__init__(parent)
        self.setWindowTitle('TreeWidget 例子')
        self.tree = QTreeWidget()
        # 设置列数
        self.tree.setColumnCount(2)
        # 设置树形控件头部的标题
        self.tree.setHeaderLabels(['Key', 'Value'])
        root = QTreeWidgetItem(self.tree)
        root.setText(0, 'root')
        root.setText(1, '0')

        child1 = QTreeWidgetItem(root)
        child1.setText(0, 'child1')
        child1.setText(1, '1')

        child2 = QTreeWidgetItem(root)
        child2.setText(0, 'child2')
        child2.setText(1, '2')

        child3 = QTreeWidgetItem(root)
        child3.setText(0, 'child3')
        child3.setText(1, '3')

        child4 = QTreeWidgetItem(child3)
        child4.setText(0, 'child4')
        child4.setText(1, '4')

        child5 = QTreeWidgetItem(child3)
        child5.setText(0, 'child5')
        child5.setText(1, '5')

        self.tree.addTopLevelItem(root)
        self.tree.clicked.connect( self.onTreeClicked )
```

```

        self.setCentralWidget(self.tree)

    def onTreeClicked(self, qmodelindex):
        item = self.tree.currentItem()
        print("key=%s ,value=%s" % (item.text(0), item.text(1)))

if __name__ == '__main__':
    app = QApplication(sys.argv)
    tree = TreeWidgetDemo()
    tree.show()
    sys.exit(app.exec_())

```

圖5-26

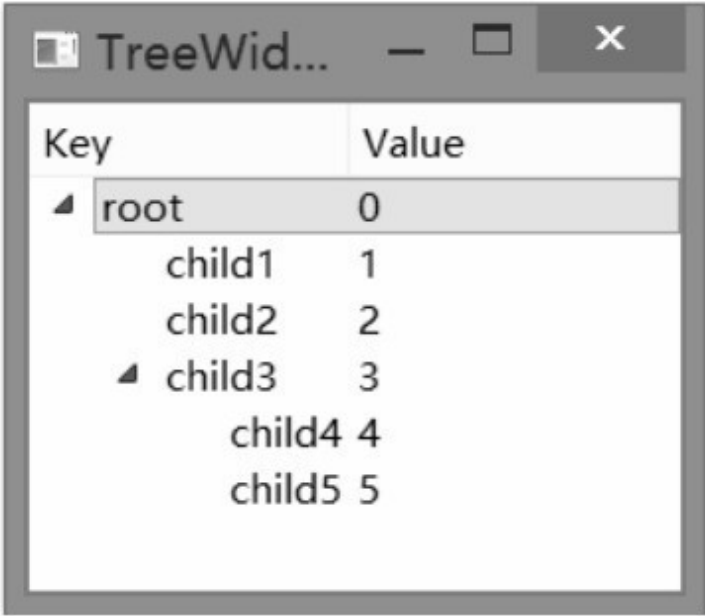


圖5-26

### 3. 實作

實作一個 `QTreeWidget` 的實例，並將其顯示在 `QTreeView` 中。在 `QTreeWidget` 中，我們需要實現 `QTreeWidget` 的 `QTreeWidget` 類，並將其顯示在 `QTreeView` 中。在 `QTreeWidget` 中，我們需要實現 `QTreeWidget` 的 `QTreeWidget` 類，並將其顯示在 `QTreeView` 中。

□□□□□PyQt5/Chapter05/qt05\_treeview.py□□□□□□□□□□

□□□□□□□□□□□□□□

```
import sys
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
if __name__ == '__main__':
    app=QApplication(sys.argv)
    # Window□□□□□□□
    model=QDirModel()
    # □□□□QTreeView□□
    tree=QTreeView()
    # □□□□□□□
    tree.setModel(model)
    tree.setWindowTitle( "QTreeView □□" )
    tree.resize(640,480)
    tree.show()
    sys.exit(app.exec_())
```

□□□□□□□□□□5-27□□□

QTreeView 例子			
Name	Size	Type	Date Modified
<div> <div>  C: </div> <div>Drive</div> <div>2017/5/10 12:49</div> </div>			
<div> <div>  AVScanner.ini </div> <div>30 byte(s) ini File</div> <div>2017/5/10 12:46</div> </div>			
<div> <div>  Intel </div> <div>File Folder</div> <div>2015/2/8 17:17</div> </div>			
<div> <div>  PerfLogs </div> <div>File Folder</div> <div>2013/8/22 23:22</div> </div>			
<div> <div>  Program Files </div> <div>File Folder</div> <div>2017/3/25 14:23</div> </div>			
<div> <div>  Program Files (x86) </div> <div>File Folder</div> <div>2017/5/10 12:49</div> </div>			
<div> <div>  Users </div> <div>File Folder</div> <div>2017/3/21 13:53</div> </div>			
<div> <div>  Windows </div> <div>File Folder</div> <div>2017/5/16 1:01</div> </div>			
<div> <div>  mfg </div> <div>File Folder</div> <div>2015/2/8 16:33</div> </div>			
<div> <div>  tmp </div> <div>File Folder</div> <div>2017/4/11 9:06</div> </div>			
<div> <div>  D: </div> <div>Drive</div> <div>2017/5/17 21:50</div> </div>			
<div> <div>  E: </div> <div>Drive</div> <div>2017/5/17 20:13</div> </div>			

图5-27

## 5.2 树形控件

树形控件是用于显示树形结构的控件，它通常用于显示文件系统、目录结构、数据库结构等。在Qt中，QTreeView是用于显示树形结构的控件，它支持多种树形结构，如目录树、文件树、数据库树等。

### 5.2.1 QTabWidget

QTabWidget是Qt中的一个控件，用于显示多个文档或视图。它通常用于显示多个文档、多个视图、多个数据表等。QTabWidget支持多种样式，如单页式、多页式、多页式等。在Qt中，QTabWidget是用于显示多个文档或视图的控件，它支持多种样式，如单页式、多页式、多页式等。

□□□□□□

QTabWidget□□□□□□□□5-13□□□

□5-13

方 法	描 述
addTab()	将一个控件添加到 Tab 控件的选项卡中
insertTab()	将一个 Tab 控件的选项卡插入到指定的位置
removeTab()	根据指定的索引删除 Tab 控件
setCurrentIndex()	设置当前可见的选项卡所在的索引
setCurrentWidget()	设置当前可见的页面
setTabBar()	设置选项卡栏的小控件
setTabPosition()	设置选项卡的位置： <ul style="list-style-type: none"><li>• QTabWidget.North，显示在页面的上方</li><li>• QTabWidget.South，显示在页面的下方</li><li>• QTabWidget.West，显示在页面的左侧</li><li>• QTabWidget.East，显示在页面的右侧</li></ul>
setTabText()	定义 Tab 选项卡的显示值

QTabWidget□□□□□□□□5-14□□□

□5-14

信 号	描 述
currentChanged	切换当前页面时发射该信号

[□□5-4 QTabWidget□□□](#)

□□□□□PyQt5/Chapter05/qt05\_QTabWidget.py□□□□PyQt 5  
□□□□□QTabWidget□□□□□□□□□□

```
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class TabDemo(QTabWidget):
    def __init__(self, parent=None):
        super(TabDemo, self).__init__(parent)
        self.tab1 = QWidget()
        self.tab2 = QWidget()
        self.tab3 = QWidget()
        self.addTab(self.tab1,"Tab 1")
        self.addTab(self.tab2,"Tab 2")
        self.addTab(self.tab3,"Tab 3")
```



```

        self.tab1UI()
        self.tab2UI()
        self.tab3UI()
        self.setWindowTitle("Tab 例子")

    def tab1UI(self):
        layout = QFormLayout()
        layout.addRow("姓名", QLineEdit())
        layout.addRow("地址", QLineEdit())
        self.setTabText(0, "联系方式")
        self.tab1.setLayout(layout)

    def tab2UI(self):
        layout = QFormLayout()
        sex = QHBoxLayout()
        sex.addWidget(QRadioButton("男"))
        sex.addWidget(QRadioButton("女"))
        layout.addRow(QLabel("性别"), sex)
        layout.addRow("生日", QLineEdit())
        self.setTabText(1, "个人详细信息")
        self.tab2.setLayout(layout)

    def tab3UI(self):
        layout = QHBoxLayout()
        layout.addWidget(QLabel("科目"))
        layout.addWidget(QCheckBox("物理"))
        layout.addWidget(QCheckBox("高数"))
        self.setTabText(2, "教育程度")
        self.tab3.setLayout(layout)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = TabDemo()
    demo.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□5-28□□5-29□□5-30□□□



图5-28



图5-29



图5-30

```

class MainWindow(QMainWindow):
    def __init__(self):
        QMainWindow.__init__(self)
        self.tab1=QWidget()

```

```

self.tab2=QWidget()
self.tab3=QWidget()
self.addTab(self.tab1,"Tab 1")
self.addTab(self.tab2,"Tab 2")
self.addTab(self.tab3,"Tab 3")
#####
self.setTabText(0,"#####")
self.setTabText(1,"#####")
self.setTabText(2,"#####")

```

## 5.2.2 QStackedWidget

QStackedWidget 是 Qt 中用于堆叠显示多个子窗口的类。它包含以下成员函数：

- QStackedWidget() 构造函数
- QStackedLayout() 构造函数
- QStackedWidget() 构造函数
- QTabWidget() 构造函数

### 5-5 QStackedWidget 例子

本例代码位于 PyQt5/Chapter05/qt05\_QStackedWidget.py 文件中。PyQt 5 中 QStackedWidget 的例子如下：

```

import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class StackedExample(QWidget):
    def __init__(self):
        super(StackedExample, self).__init__()
        self.setGeometry(300, 50, 10, 10)
        self.setWindowTitle('StackedWidget 例子')

        self.leftlist = QListWidget ()

```

```

self.leftlist.insertItem (0, '联系方式' )
self.leftlist.insertItem (1, '个人信息' )
self.leftlist.insertItem (2, '教育程度' )
self.stack1 = QWidget()
self.stack2 = QWidget()
self.stack3 = QWidget()
self.stack1UI()
self.stack2UI()
self.stack3UI()
self.Stack = QStackedWidget (self)
self.Stack.addWidget(self.stack1)
self.Stack.addWidget(self.stack2)
self.Stack.addWidget(self.stack3)
hbox = QHBoxLayout(self)
hbox.addWidget(self.leftlist)
hbox.addWidget(self.Stack)
self.setLayout(hbox)
self.leftlist.currentRowChanged.connect(self.display)

def stack1UI(self):
    layout=QFormLayout()
    layout.addRow("姓名",QLineEdit())
    layout.addRow("地址",QLineEdit())
    self.stack1.setLayout(layout)

def stack2UI(self):
    layout = QFormLayout()
    sex = QHBoxLayout()
    sex.addWidget(QRadioButton("男"))
    sex.addWidget(QRadioButton("女"))
    layout.addRow(QLabel("性别"),sex)
    layout.addRow("生日",QLineEdit())
    self.stack2.setLayout(layout)

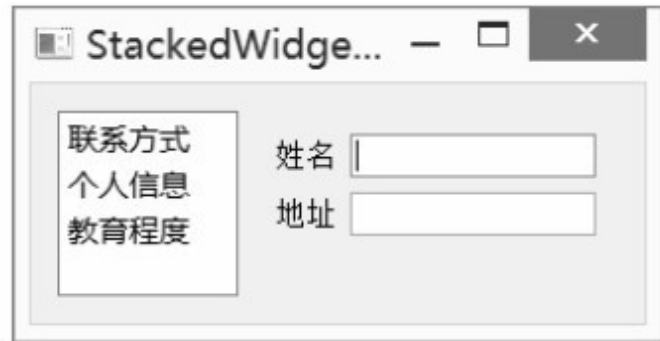
def stack3UI(self):
    layout=QHBoxLayout()
    layout.addWidget(QLabel("科目"))
    layout.addWidget(QCheckBox("物理"))
    layout.addWidget(QCheckBox("高数"))
    self.stack3.setLayout(layout)

def display(self,i):
    self.Stack.setCurrentIndex(i)

```

```
if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = StackedExample()
    demo.show()
    sys.exit(app.exec_())
```

□□□□□□□□□□5-31□□5-32□□5-33□□□



□5-31



□5-32



□5-33

```

class StackWidget(QWidget):
    def __init__(self):
        self.Stack=QStackedWidget (self)
        self.stack1=QWidget()
        self.stack2=QWidget()
        self.stack3=QWidget()
        self.Stack.addWidget(self.stack1)
        self.Stack.addWidget(self.stack2)
        self.Stack.addWidget(self.stack3)

class LeftListWidget(QListWidget):
    def __init__(self):
        self.leftlist=QListWidget ()
        self.leftlist.insertItem (0,'Item 0')
        self.leftlist.insertItem (1,'Item 1')
        self.leftlist.insertItem (2,'Item 2')
        self.leftlist.currentRowChanged.connect(self.display)

    def display(self,i):
        self.Stack.setCurrentIndex(i)

```

### **5.2.3 QDockWidget**

QDockWidget 是 QMainWindow 的子类，它用于创建dock widget。QMainWindow 是 QMainWindow 的子类，它用于创建 QMainWindow 窗口。5-34

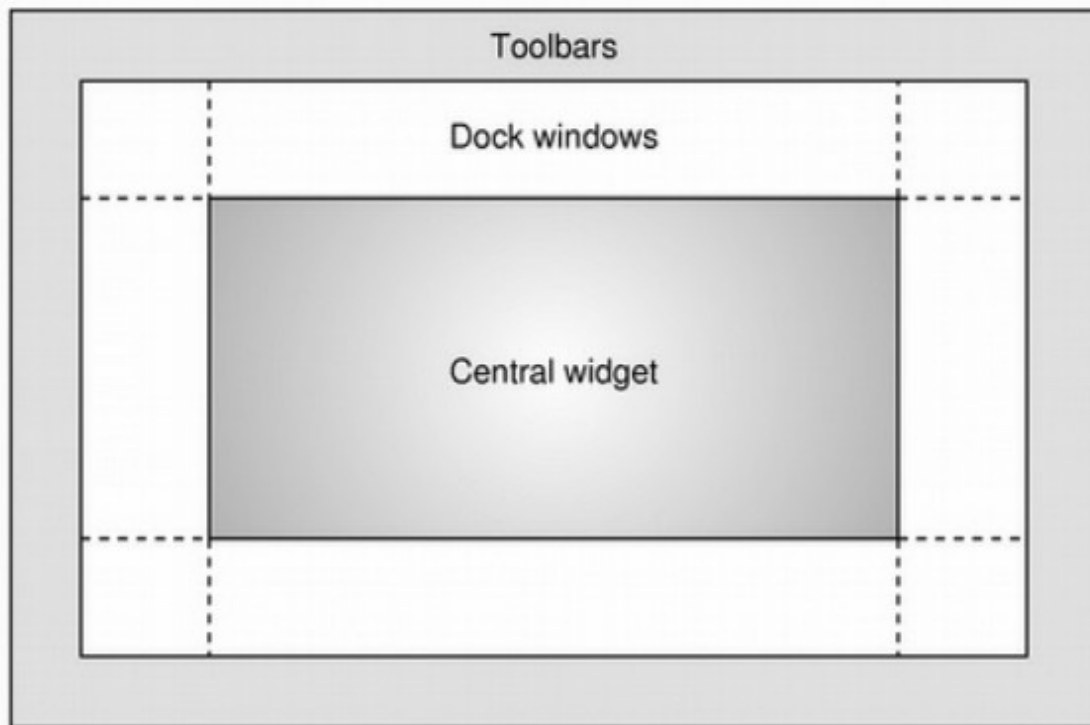


图5-34

QDockWidget 和 QMainWindow 的关系图如下所示。QDockWidget 是 QMainWindow 的子类，它继承自 QMainWindow。

图5-15

方 法	描 述
setWidget()	在 Dock 窗口区域设置 QWidget
setFloating()	设置 Dock 窗口是否可以浮动，如果设置为 True，则表示可以浮动
setAllowedAreas()	设置窗口可以停靠的区域： <ul style="list-style-type: none"> <li>• LeftDockWidgetArea，左边停靠区域</li> <li>• RightDockWidgetArea，右边停靠区域</li> <li>• TopDockWidgetArea，顶部停靠区域</li> <li>• BottomDockWidgetArea，底部停靠区域</li> <li>• NoDockWidgetArea，不显示 Widget</li> </ul>
setFeatures()	设置停靠窗口的功能属性： <ul style="list-style-type: none"> <li>• DockWidgetClosable，可关闭</li> <li>• DockWidgetMovable，可移动</li> <li>• DockWidgetFloatable，可漂浮</li> <li>• DockWidgetVerticalTitleBar，在左边显示垂直的标签栏</li> <li>• AllDockWidgetFeatures，具有前三种属性的所有功能</li> <li>• NoDockWidgetFeatures，无法关闭，不能移动，不能漂浮</li> </ul>

## [5-6 QDockWidget](#)

[PyQt5/Chapter05/qt05\\_QDockWidget.py](#)
[PyQt5](#)
[QDockWidget](#)



```

import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class DockDemo(QMainWindow):
    def __init__(self, parent=None):
        super(DockDemo, self).__init__(parent)
        layout = QHBoxLayout()
        bar=self.menuBar()
        file=bar.addMenu("File")
        file.addAction("New")
        file.addAction("save")
        file.addAction("quit")
        self.items = QDockWidget("Dockable", self)
        self.listWidget = QListWidget()
        self.listWidget.addItem("item1")
        self.listWidget.addItem("item2")
        self.listWidget.addItem("item3")
        self.items.setWidget(self.listWidget)
        self.items.setFloating(False)
        self.setCentralWidget(QTextEdit())
        self.addDockWidget(Qt.RightDockWidgetArea, self.items)
        self.setLayout(layout)
        self.setWindowTitle("Dock 例子")

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = DockDemo()
    demo.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□5-35□□□



图5-35

```

#include <QtWidgets/QMainWindow>
#include <QtWidgets/QTextEdit>
#include <QtWidgets/QDockWidget>
#include <QtWidgets/QListWidget>

using namespace Qt;

int main()
{
    QMainWindow mainWindow;
    QTextEdit textEdit;
    mainWindow.setCentralWidget(textEdit);

    QDockWidget items;
    mainWindow.items() = items;

    QListWidget listWidget;
    listWidget.addItem("item1");
    listWidget.addItem("item2");
    listWidget.addItem("item3");
    items.setWidget(listWidget);

    mainWindow.addDockWidget(Qt::RightDockWidgetArea, items);
}

```

## 5.2.4 停靠窗口

GUI 应用程序中，MDI 是一种常用的窗口管理方式。在 MDI 应用中，主窗口包含一个或多个子窗口。子窗口可以具有自己的标题栏、菜单和工具栏。MDI 应用通常用于需要同时打开和编辑多个文档的场景。

在 Qt 中，MDI 应用通常使用 `QMainWindow` 类来实现。主窗口包含一个 `QMDIArea` 对象，用于管理子窗口。子窗口是 `QMDISubWindow` 对象的实例。主窗口和子窗口之间的交互通过 `QMDIArea` 和 `QMDISubWindow` 类的方法来实现。

MDI (Multiple Document Interface) 是一种窗口管理方式，允许在一个主窗口中同时打开多个子窗口。每个子窗口可以独立地显示和编辑文档。MDI 应用通常用于需要同时处理多个文档的场景。

`QMDIArea` 是 `QMainWindow` 的一个子类，用于管理子窗口。它提供了添加、删除和激活子窗口的方法。子窗口是 `QMDISubWindow` 对象的实例，它继承自 `QWidget`。主窗口和子窗口之间的交互通过 `QMDIArea` 和 `QMDISubWindow` 类的方法来实现。

`QMDIArea` 和 `QMDISubWindow` 类的方法在图 5-16 中给出。

图 5-16

方 法	描 述
<code>addSubWindow()</code>	将一个小控件添加在 MDI 区域作为一个新的子窗口
<code>removeSubWindow()</code>	删除一个子窗口中的小控件

续

方 法	描 述
<code>setActiveSubWindow()</code>	激活一个子窗口
<code>cascadeSubWindows()</code>	安排子窗口在 MDI 区域级联显示
<code>tileSubWindows()</code>	安排子窗口在 MDI 区域平铺显示
<code>closeActiveSubWindow()</code>	关闭活动的子窗口
<code>subWindowList()</code>	返回 MDI 区域的子窗口列表
<code>setWidget()</code>	设置一个小控件作为 <code>QMDISubwindow</code> 实例对象的内部控件

图 5-7 窗口管理

在 PyQt5 中，MDI 应用通常使用 `QMainWindow` 类来实现。主窗口包含一个 `QMDIArea` 对象，用于管理子窗口。子窗口是 `QMDISubWindow` 对象的实例。主窗口和子窗口之间的交互通过 `QMDIArea` 和 `QMDISubWindow` 类的方法来实现。

```
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class MainWindow(QMainWindow):
    count=0
    def __init__(self, parent=None):
        super(MainWindow, self).__init__(parent)
        self.mdi = QMdiArea()
        self.setCentralWidget(self.mdi)
        bar=self.menuBar()
        file=bar.addMenu("File")
        file.addAction("New")
        file.addAction("cascade")
        file.addAction("Tiled")
        file.triggered[QAction].connect(self.windowaction)
        self.setWindowTitle("MDI demo")

    def windowaction(self, q):
        print( "triggered")

        if q.text()=="New":
            MainWindow.count=MainWindow.count+1
            sub=QMdiSubWindow()
            sub.setWidget(QTextEdit())
```

```

        sub.setWindowTitle("subwindow"+str(MainWindow.count))
        self.mdi.addSubWindow(sub)
        sub.show()
    if q.text()=="cascade":
        self.mdi.cascadeSubWindows()
    if q.text()=="Tiled":
        self.mdi.tileSubWindows()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = MainWindow()
    demo.show()
    sys.exit(app.exec_())

```

5-36



5-36

QMainWindow MidArea

```

self.mdi=QMdiArea()
self.setCentralWidget(self.mdi)
bar=self.menuBar()
file=bar.addMenu("File")
file.addAction("New")
file.addAction("cascade")
file.addAction("Tiled")
file.triggered.connect(self.windowaction)
file.triggered[QAction].connect(self.windowaction)
file.addAction("New")
MDI
MDI
MDI
MainWindow.count=MainWindow.count+1
sub=QMdiSubWindow()
sub.setWidget(QTextEdit())
sub.setWindowTitle("subwindow"+str(MainWindow.count))
self.mdi.addSubWindow(sub)
sub.show()
file.addAction("cascade")
file.addAction("Tiled")
if q.text()=="cascade":
    self.mdi.cascadeSubWindows()
if q.text()=="Tiled":
    self.mdi.tileSubWindows()

```

## 5.2.5 QScrollBar

在Qt5中，QScrollBar类用于创建滚动条。它继承自QAbstractSlider，并实现了QSlider接口。QScrollBar类提供了许多属性，用于控制滚动条的外观和行为。例如，可以通过setRange()方法设置滚动条的取值范围，通过setPageStep()方法设置滚动条的页面步长。此外，QScrollBar类还发射了valueChanged()和sliderMoved()信号，用于响应滚动条值的变化。

QScrollBar类成员函数5-17

5-17

信 号	含 义
valueChanged	当滑动条的值改变时发射此信号
sliderMoved	当用户拖动滑块时发射此信号

5-8 QScrollBar

在PyQt5/Chapter05/qt05\_QScrollBar.py文件中，我们使用PyQt 5的QScrollBar类来创建一个滚动条。以下代码展示了如何导入必要的模块并创建滚动条实例。

```
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
```

```

class Example(QWidget):
    def __init__(self):
        super(Example, self).__init__()
        self.initUI()

    def initUI(self):
        hbox = QHBoxLayout( )
        self.l1 = QLabel("拖动滑块改变颜色")
        self.l1.setFont(QFont("Arial",16))
        hbox.addWidget(self.l1)
        self.s1 = QScrollBar()
        self.s1.setMaximum(255)
        self.s1.sliderMoved.connect(self.sliderval)
        self.s2 = QScrollBar()
        self.s2.setMaximum(255)
        self.s2.sliderMoved.connect(self.sliderval)
        self.s3 = QScrollBar()
        self.s3.setMaximum(255)
        self.s3.sliderMoved.connect(self.sliderval)
        hbox.addWidget(self.s1)
        hbox.addWidget(self.s2)
        hbox.addWidget(self.s3)
        self.setGeometry(300, 300, 300, 200)
        self.setWindowTitle('QScrollBar 例子')
        self.setLayout( hbox )

    def sliderval(self):
        print( self.s1.value(),self.s2.value(), self.s3.value() )
        palette = QPalette()
        c=QColor(self.s1.value(),self.s2.value(),
self.s3.value(),255)
        palette.setColor(QPalette.Foreground,c)
        self.l1.setPalette(palette)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = Example()
    demo.show()
    sys.exit(app.exec_())

```



图5-37



图5-37

代码如下：  
通过调用 `sliderMoved` 信号和 `sliderval()` 槽函数，  
`self.s1.sliderMoved.connect(self.sliderval)`

## 5.3 多线程

在 GUI 应用程序中，通常需要将耗时的操作放在后台线程中执行，以避免阻塞主线程。在 Windows 平台上，可以使用 `QTimer` 和 `QThread` 类来实现多线程编程。

### 5.3.1 QTimer

在CPU空闲时，QtTimer的timeout信号将不会发出。因此，在启动QtTimer时，应设置一个合理的timeout值，以确保在CPU空闲时，QtTimer能够正常工作。

图5-18展示了QtTimer的启动和停止过程。

图5-18

方 法	描 述
start(milliseconds)	启动或重新启动定时器，时间间隔为毫秒。如果定时器已经运行，它将被停止并重新启动。如果 singleShot 信号为真，定时器将仅被激活一次
Stop()	停止定时器

QtTimer的启动和停止过程如图5-19所示。

图5-19

信 号	描 述
singleShot	在给定的时间间隔后调用一个槽函数时发射此信号
timeout	当定时器超时时发射此信号

```

class QTimer:
    def __init__(self, timeout=1000):
        self.timeout = timeout
        self.start(2000)

    def start(self, timeout):
        self.timeout = timeout
        self.timer = QTimer(self)
        self.timer.timeout.connect(self.operate)
        self.timer.start(2000)

    def operate(self):
        # 这里可以放置需要执行的操作
        # 这里可以放置需要执行的操作
        self.timer.start(2000)
    
```

## 1. 실행

파일명 PyQt5/Chapter05/qt05\_timer01.py 실행시간 10

실행결과

```
from PyQt5.QtWidgets import QWidget, QPushButton ,
QApplication ,QListWidget, QGridLayout , QLabel
from PyQt5.QtCore import QTimer ,QDateTime
import sys

class WinForm(QWidget):

    def __init__(self,parent=None):
        super(WinForm,self).__init__(parent)
```

```

self.setWindowTitle("QTimer demo")
self.listFiles = QListWidget()
self.label = QLabel('显示当前时间')
self.startBtn = QPushButton('开始')
self.endBtn = QPushButton('结束')
layout = QGridLayout(self)

# 初始化一个定时器
self.timer = QTimer(self)
# showTime() 方法
self.timer.timeout.connect(self.showTime)

layout.addWidget(self.label, 0, 0, 1, 2)
layout.addWidget(self.startBtn, 1, 0)
layout.addWidget(self.endBtn, 1, 1)

self.startBtn.clicked.connect(self.startTimer)
self.endBtn.clicked.connect(self.endTimer)

self.setLayout(layout)

def showTime(self):
    # 获取系统现在的时间
    time = QDateTime.currentDateTime()
    # 设置系统时间显示格式
    timeDisplay = time.toString("yyyy-MM-dd hh:mm:ss dddd");
    # 在标签上显示时间
    self.label.setText(timeDisplay)

def startTimer(self):
    # 设置时间间隔并启动定时器
    self.timer.start(1000)
    self.startBtn.setEnabled(False)
    self.endBtn.setEnabled(True)

def endTimer(self):
    self.timer.stop()
    self.startBtn.setEnabled(True)
    self.endBtn.setEnabled(False)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = WinForm()

```

```
form.show()
sys.exit(app.exec_())
```

图5-38

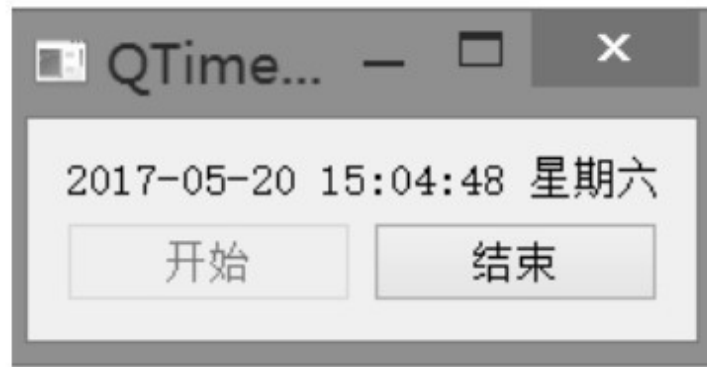


图5-38

代码如下

```
from PyQt5.QtCore import QTimer, QDateTime
from PyQt5.QtGui import QLabel
from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget, QVBoxLayout, QPushButton

class TimeWidget(QWidget):
    def __init__(self):
        super().__init__()
        self.timer = QTimer(self)
        self.timer.timeout.connect(self.showTime)
        self.label = QLabel()
        self.timer.start(1000)

    def showTime(self):
        # 获取当前日期时间
        time = QDateTime.currentDateTime()
        # 格式化日期时间
        timeDisplay = time.toString("yyyy-MM-dd hh:mm:ss dddd");
        # 设置标签文本
        self.label.setText( timeDisplay )

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = QMainWindow()
    widget = TimeWidget()
    window.setCentralWidget(widget)
    window.show()
    sys.exit(app.exec_())
```

```

self.timer.start(1000)
self.startBtn.setEnabled(False)
self.endBtn.setEnabled(True)
print("开始倒计时")
self.timer.stop()
self.startBtn.setEnabled(True)
self.endBtn.setEnabled(False)

```

## 2. 例2

下面代码PyQt5/Chapter05/qt05\_timer02.py实现一个倒计时器，倒计时10秒后自动退出。

```

import sys
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
from PyQt5.QtCore import *

if __name__ == '__main__':
    app = QApplication(sys.argv)
    label = QLabel("<font color=red size=128><b>Hello PyQt, 窗口会在 10  
秒后消失! </b></font>")
    # 无边框窗口
    label.setWindowFlags(Qt.SplashScreen|Qt.FramelessWindowHint)
    label.show()

    # 设置 10 秒后自动退出
    QTimer.singleShot(10000, app.quit)
    sys.exit(app.exec_())

```

运行结果如图5-39所示。

Hello PyQt, 窗口会在10秒后消失!

```

    显示
    每隔1000毫秒显示一次
    label.setWindowFlags(Qt.SplashScreen|Qt.FramelessWindowHint)
    QTimer.singleShot(10000,app.quit)

```

### 5.3.2 QThread

QThread 是 Qt 库中提供的一个类，PyQt 中也有 QThread 类，  
 使用 QThread 类可以实现多线程编程。  
 使用 QThread 类实现多线程编程的步骤如下：  
 1. 继承 QThread 类并实现 run() 方法。  
 2. 创建 QThread 对象并调用 start() 方法启动线程。

```

class Thread(QThread):
    def __init__(self):
        super(Thread,self).__init__()

```

```

    def run(self):
        #线程相关代码
        pass

```

```

#创建线程对象
thread=Thread()
thread.start()

#使用PyQt库中的QThread类——QThread类是QThread类的子类
QThread类中的run()方法是空方法
使用QThread类实现多线程编程的步骤如下
1. 继承Thread类并实现start()方法
2. 创建Thread对象并调用run()方法启动线程

```

run()run()QThread started finished

1.QThread
QThread5-20

5-20

方 法	描 述
start()	启动线程
wait()	阻止线程，直到满足如下条件之一： <ul style="list-style-type: none"><li>与此 QThread 对象关联的线程已完成执行（即从 run()返回时）。如果线程完成执行，此函数将返回 True；如果线程尚未启动，此函数也返回 True</li><li>等待时间的单位是毫秒。如果时间是 ULONG_MAX（默认值），则等待，永远不会超时（线程必须从 run()返回）；如果等待超时，此函数将返回 False</li></ul>
sleep()	强制当前线程睡眠秒秒。

QThread5-21

5-21

信 号	描 述
started	在开始执行 run()函数之前，从相关线程发射此信号
finished	当程序完成业务逻辑时，从相关线程发射此信号

2.QThread

PyQt5/Chapter05/qt05\_thread03.py
1010



```
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
import sys

class MainWidget(QWidget):
    def __init__(self, parent=None):
        super(MainWidget, self).__init__(parent)
        self.setWindowTitle("QThread 例子")
        self.thread = Worker()
        self.listFiles = QListWidget()
        self.btnStart = QPushButton('开始')
        layout = QGridLayout(self)
        layout.addWidget(self.listFiles, 0, 0, 1, 2)
        layout.addWidget(self.btnStart, 1, 1)
        self.btnStart.clicked.connect(self.slotStart)
        self.thread.sinOut.connect(self.slotAdd)

    def slotAdd(self, file_inf):
        self.listFiles.addItem(file_inf)

    def slotStart(self):
        self.btnStart.setEnabled(False)
        self.thread.start()

class Worker(QThread):
    sinOut = pyqtSignal(str)

    def __init__(self, parent=None):
        super(Worker, self).__init__(parent)
        self.working = True
        self.num = 0

    def __del__(self):
        self.working = False
        self.wait()
```

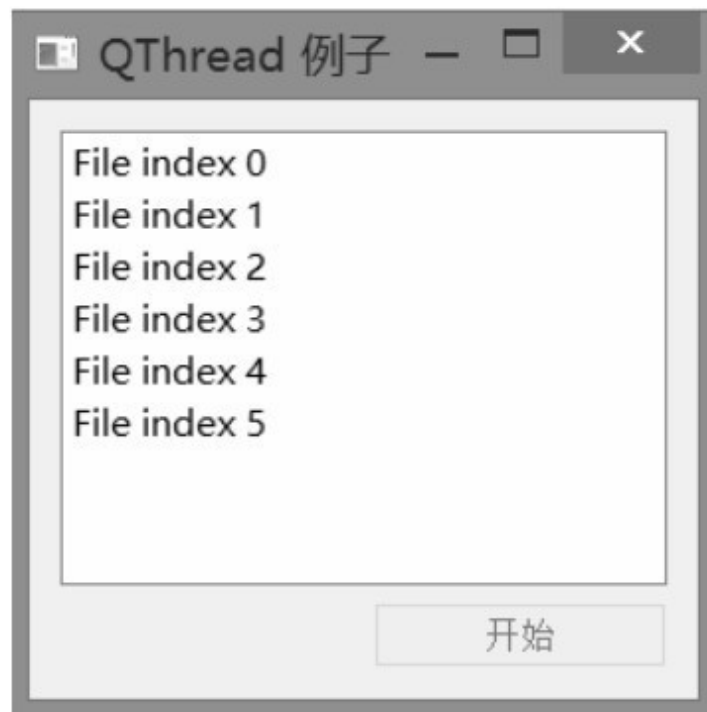
```

def run(self):
    while self.working == True:
        file_str = 'File index {0}'.format(self.num)
        self.num += 1
        # 发射信号
        self.sinOut.emit(file_str)
        # 线程休眠 2 秒
        self.sleep(2)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    demo = MainWidget()
    demo.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□5-40□□□



□5-40

□□□□□

```

        """
        self.listFile=QListWidget()
        self.btnStart=QPushButton('开始')
        layout=QGridLayout(self)
        layout.addWidget(self.listFile,0,0,1,2)
        layout.addWidget(self.btnStart,1,1)
        self.clicked.connect(self.slotStart)"""
        self.btnStart.clicked.connect( self.slotStart )
        def slotStart(self):
            self.btnStart.setEnabled(False)
            self.thread.start()

        self.sinOut.connect(self.slotAdd)"""
        self.thread.sinOut.connect(self.slotAdd)
        def slotAdd(self,file_inf):
            self.listFile.addItem(file_inf)
        self.QThread.run()

```



```
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

global sec
sec=0

def setTime():
    global sec
    sec+=1
    # LED 显示数字+1
    lcdNumber.display(sec)

def work():
    # 计时器每秒计数
    timer.start(1000)
    for i in range(2000000000):
        pass

    timer.stop()

if __name__ == "__main__":
    app = QApplication(sys.argv)
    top = QWidget()
    top.resize(300,120)

    # 垂直布局类 QVBoxLayout
    layout = QVBoxLayout(top)
    # 添加一个显示面板
    lcdNumber = QLCDNumber()
    layout.addWidget(lcdNumber)
    button=QPushButton("测试")
    layout.addWidget(button)

    timer = QTimer()
    # 每次计时结束，触发 setTime
    timer.timeout.connect(setTime)
    button.clicked.connect(work)

    top.show()
    sys.exit(app.exec_())
```

5-41



□5-41

00000000000000000000 LCD 00000000000000000000  
 00“00”00000000000000000000000000 2 000 000 000 00000000  
 000 LCD  
 000  
 00000000000000000000000000000000“00”0000000000000000  
 0000000000000000000000000000000000

PyQt UI QApplication.exec() UI QtThread

## 5-9 UI

PyQt5/Chapter05/qt05\_thread02.py UI

```
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

global sec
sec=0

class WorkThread(QThread):
    trigger = pyqtSignal()
    def __init__(self):
        super(WorkThread,self).__init__()
```

```

def run(self):
    for i in range(2000000000):
        pass

    # 循环完毕后发射信号
    self.trigger.emit()

def countTime():
    global sec
    sec += 1
    # LED显示数字+1
    lcdNumber.display(sec)

def work():
    # 计时器每秒计数
    timer.start(1000)
    # 计时开始
    workThread.start()
    # 当获得循环完毕的信号时, 停止计数
    workThread.trigger.connect(timeStop)

def timeStop():
    timer.stop()
    print("运行结束用时", lcdNumber.value())
    global sec
    sec=0

if __name__ == "__main__":
    app = QApplication(sys.argv)
    top = QWidget()
    top.resize(300,120)

    # 垂直布局类 QVBoxLayout
    layout = QVBoxLayout(top)
    # 添加一个显示面板
    lcdNumber = QLCDNumber()
    layout.addWidget(lcdNumber)
    button = QPushButton("测试")
    layout.addWidget(button)

    timer = QTimer()
    workThread = WorkThread()

```



```

button.clicked.connect(work)
# 每次计时结束，触发 countTime
timer.timeout.connect(countTime)

top.show()
sys.exit(app.exec_())

```

在 Qt 中，我们使用 `WorkerThread` 类来创建线程。该类的 `run()` 方法是我们需要实现的方法，它将在线程启动时调用。在 `run()` 方法中，我们调用 `work()` 方法，该方法将执行我们想要在线程中运行的代码。最后，我们调用 `workThread.start()` 方法来启动线程。

图 5-42 所示



图 5-42

### 5.3.3 多线程

PyQt 提供了 `processEvents()` 方法，用于处理事件。在 PyQt 中，我们使用 `QApplication` 类来创建应用程序。在 `QApplication` 类中，我们调用 `processEvents()` 方法来处理事件。

```

QApplication.processEvents()
QApplication.processEvents()
    PyQt5/Chapter05/qt05_freshUi.py

```

```
from PyQt5.QtWidgets import QWidget, QPushButton,
QApplication, QListWidget, QGridLayout
```

```

import sys
import time

class WinForm(QWidget):

    def __init__(self, parent=None):
        super(WinForm, self).__init__(parent)
        self.setWindowTitle("实时刷新页面例子")
        self.listFiles = QListWidget()
        self.btnStart = QPushButton('开始')
        layout = QGridLayout(self)
        layout.addWidget(self.listFiles, 0, 0, 1, 2)
        layout.addWidget(self.btnStart, 1, 1)
        self.btnStart.clicked.connect(self.slotAdd)
        self.setLayout(layout)

    def slotAdd(self):
        for n in range(10):
            str_n = 'File index {0}'.format(n)
            self.listFiles.addItem(str_n)
            QApplication.processEvents()
            time.sleep(1)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = WinForm()
    form.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□5-43□□□

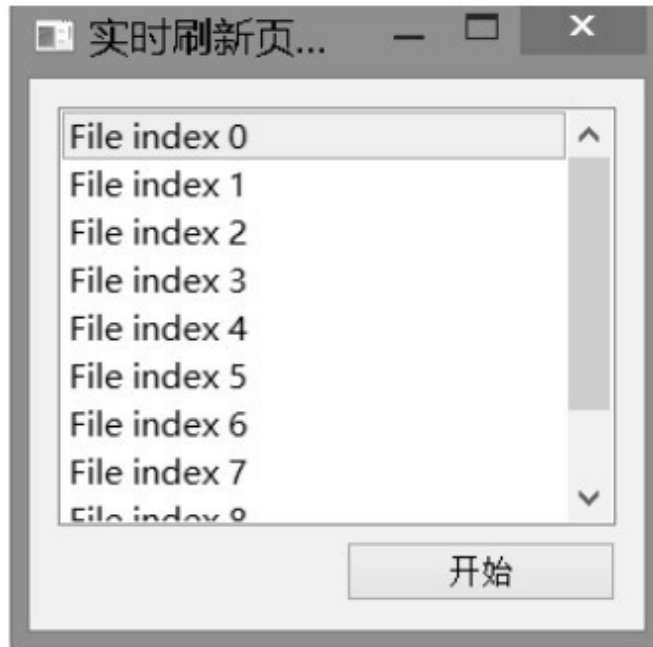


图5-43

## 5.4 网络

PyQt 5 使用 `QWebEngineView` 来加载 HTML 页面。它基于 `QWebView` 和 `QWebEngineView`。Chromium 是 Qt 5 的默认渲染引擎。

Qt 5 使用 `WebKit` 渲染引擎。WebEngine 是 Qt 5 的默认渲染引擎。Chromium 是 Qt 5 的默认渲染引擎。WebEngine 是 Qt 5 的默认渲染引擎。Chromium 是 Qt 5 的默认渲染引擎。HTML 5

PyQt 5 使用 `PyQt5.QtWebKitWidgets.QWebEngineView` 来加载 HTML 页面。



## QWebEngineView 5-22

5-22

方 法	描 述
load(QUrl url)	加载指定的 URL 并显示
setHtml(QString &html)	将网页视图的内容设置为指定的 HTML 内容

QWebEngineView 使用 load() 方法加载 Web 内容。使用 HTTP GET 方法加载 Web 内容。Web 内容可以是 HTML 文档、XML 文档、JSON 文档、JavaScript 文档、CSS 文档、SVG 文档、PDF 文档、视频、音频、图像、动画、游戏、应用程序等。

```
view=QWebEngineView()
view.load(QUrl('http://www.cnblogs.com/wangshuo1/'))
view.show()
QWebEngineView 使用 setHtml() 方法加载 Web 内容
```

## 5-10 使用 Web 内容

在 PyQt5/Chapter05/qt05\_webview01.py 中使用 QWebEngineView 加载 Web 内容。

```

from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from PyQt5.QtWebEngineWidgets import *
import sys

class MainWindow(QMainWindow):

    def __init__(self ):
        super(QMainWindow, self).__init__()
        self.setWindowTitle('打开外部网页例子')
        self.setGeometry(5, 30, 1355, 730)
        self.browser = QWebEngineView()
        # 加载外部的 Web 页面
        self.browser.load(QUrl('http://www.cnblogs.com/wangshuo1'))
        self.setCentralWidget(self.browser)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = MainWindow()
    win.show()
    app.exec_()

```

□□□□□□□□□□5-44□□□



5-44

## 5-11 PyQt5/Chapter05/qt05\_webview02.py

PyQt5/Chapter05/qt05\_webview02.py  
QWebView 类是 QtWebKit 模块的一部分，用于在 Qt 应用程序中嵌入 Web 内容。它提供了与 QtWebKit 引擎的接口，用于加载和显示网页内容。

```

from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from PyQt5.QtWebEngineWidgets import *
import sys

class MainWindow(QMainWindow):

    def __init__(self ):
        super(QMainWindow, self).__init__()
        self.setWindowTitle('加载并显示本地页面例子')
        self.setGeometry(5, 30, 555, 330)
        self.browser = QWebEngineView()
        # 加载本地页面
        url = r'E:/quant/PyQt5/Chapter05/index.html'
        self.browser.load( QUrl( url ))
        self.setCentralWidget(self.browser)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = MainWindow()
    win.show()
    sys.exit(app.exec_())

```

index.html HTML 5

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title></title>
</head>
<body>
    <h1>Hello PyQt5</h1>
    <h1>Hello PyQt5</h1>
    <h1>hello PyQt5</h1>

```



```

<h1>hello PyQt5</h1>
<h1>hello PyQt5</h1>
<h1>Hello PyQt5</h1>
</body>
</html>

```

图5-45



图5-45

## 图5-12 加载并显示本地HTML

图5-12 加载并显示本地HTML

图5-12 展示了 PyQt5/Chapter05/qt05\_webview03.py 文件中的 QWebEngineView 加载并显示本地 HTML 页面的例子。图5-11 显示了 index.html 文件中的 PyQt5 字样。

```
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from PyQt5.QtWebEngineWidgets import *
import sys

class MainWindow(QMainWindow):

    def __init__(self ):
        super(QMainWindow, self).__init__()
        self.setWindowTitle('加载并显示本地页面例子')
        self.setGeometry(5, 30, 1355, 730)
```

```

        self.browser = QWebEngineView()
        # 加载 HTML 代码
        self.browser = QWebEngineView()
        self.browser.setHtml(''
        <!DOCTYPE html>
        <html>
            <head>
                <meta charset="UTF-8">
                <title></title>
            </head>
            <body>
                <h1>Hello PyQt5</h1>
                <h1>Hello PyQt5</h1>
        <h1>hello PyQt5</h1>
        <h1>hello PyQt5</h1>
        <h1>hello PyQt5</h1>
        <h1>Hello PyQt5</h1>

            </body>
        </html>

        ''
    )

    self.setCentralWidget(self.browser)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = MainWindow()
    win.show()
    sys.exit(app.exec_())

```

QtWebEngineView 5-45 版本

Qt

QtWebEngineView 的 setHtml() 方法加载 HTML 代码

不支持 JavaScript 且 2MB 限制

QTBUG-53414 QtWebEngineView fails to load huge page

https://bugreports.qt.io/browse/QTBUG-53414  
Bug

## 5-13 PyQtJavaScript

QWebEnginePage runJavaScript(str,Callable)  
PyQt HTML/JavaScript Python  
HTML/JavaScript PyQt JavaScript

## QWebEnginePag.runJavaScript(str,Callable)

PyQt5/Chapter05/qt502\_webviewJs01.py

11

```
from PyQt5.QtWidgets import QApplication , QWidget , QVBoxLayout ,
QPushButton
from PyQt5.QtWebEngineWidgets import QWebEngineView
import sys

# 创建一个应用实例
app = QApplication(sys.argv)
win = QWidget()
win.setWindowTitle('Web 页面中的 JavaScript 与 QWebEngineView 交互例子')

# 创建一个垂直布局器
layout = QVBoxLayout()
win.setLayout(layout)

# 创建一个 QWebEngineView 对象
view = QWebEngineView()
view.setHtml(''
<html>
<head>
<title>A Demo Page</title>

<script language="javascript">
    # 获得输入的姓名, 然后在页面中显示提交按钮
    function completeAndReturnName() {
        var fname = document.getElementById('fname').value;
        var lname = document.getElementById('lname').value;
        var full = fname + ' ' + lname;

        document.getElementById('fullname').value = full;
        document.getElementById('submit-btn').style.display =
'block';
```

```

        return full;
    }
</script>
</head>

<body>
<form>
<label for="fname">First name:</label>
<input type="text" name="fname" id="fname"></input>
<br />
<label for="lname">Last name:</label>
<input type="text" name="lname" id="lname"></input>
<br />
<label for="fullname">Full name:</label>
<input disabled type="text" name="fullname" id="fullname"></input>
<br />
<input style="display: none;" type="submit" id="submit-btn"></input>
</form>
</body>
</html>
'''

# 创建一个按钮用于调用 JavaScript 代码
button = QPushButton('设置全名')

def js_callback(result):
    print(result)

def complete_name():
    view.page().runJavaScript('completeAndReturnName();', js_callback)

# 按钮连接'complete_name'槽函数，当单击按钮时会触发信号
button.clicked.connect(complete_name)

# 把QWebEngineView控件和按钮控件加载到 layout 布局中
layout.addWidget(view)
layout.addWidget(button)

# 显示窗口和运行
win.show()
sys.exit(app.exec_())

```

图5-46图5-47图5-48



图5-46

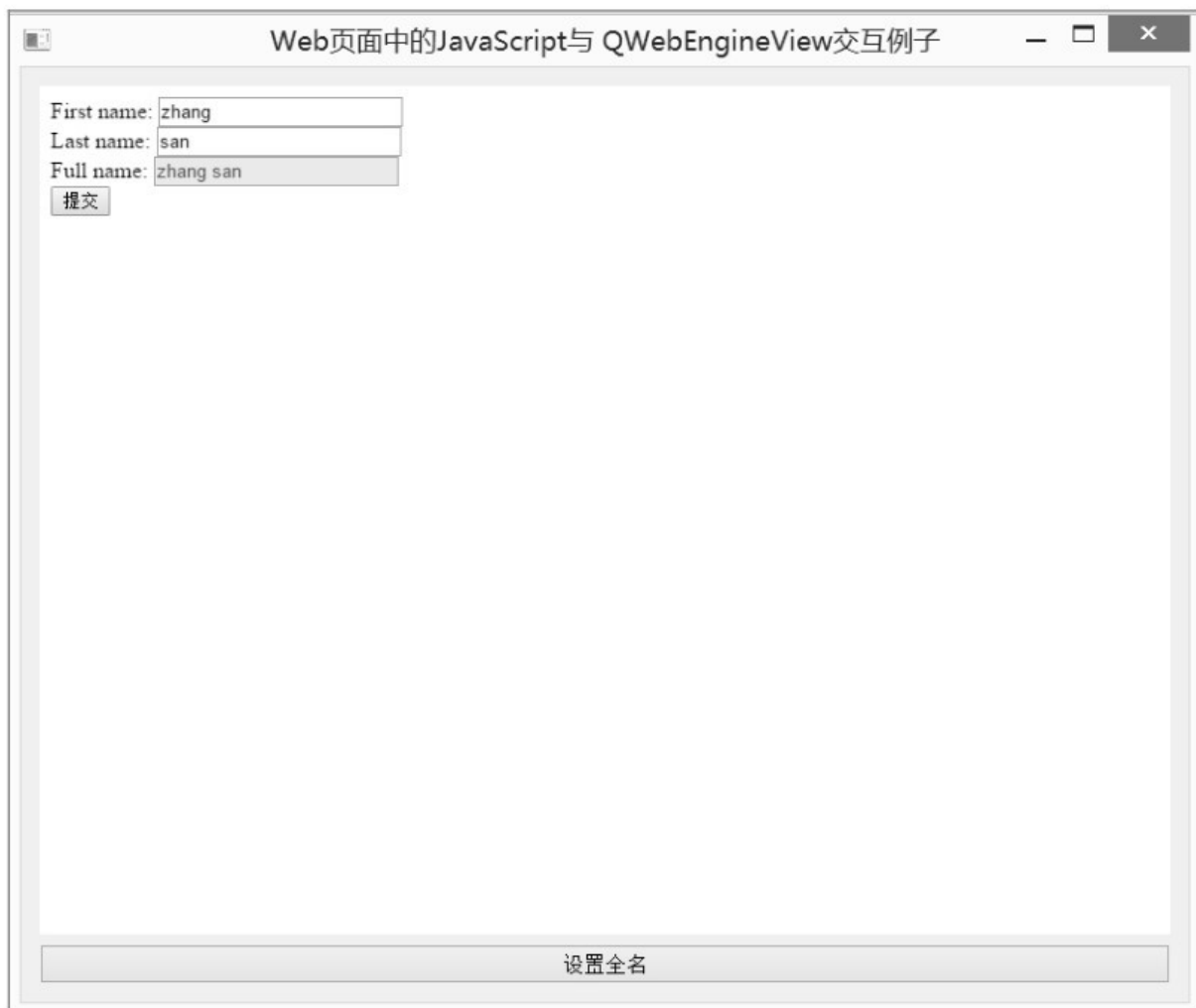


图5-47



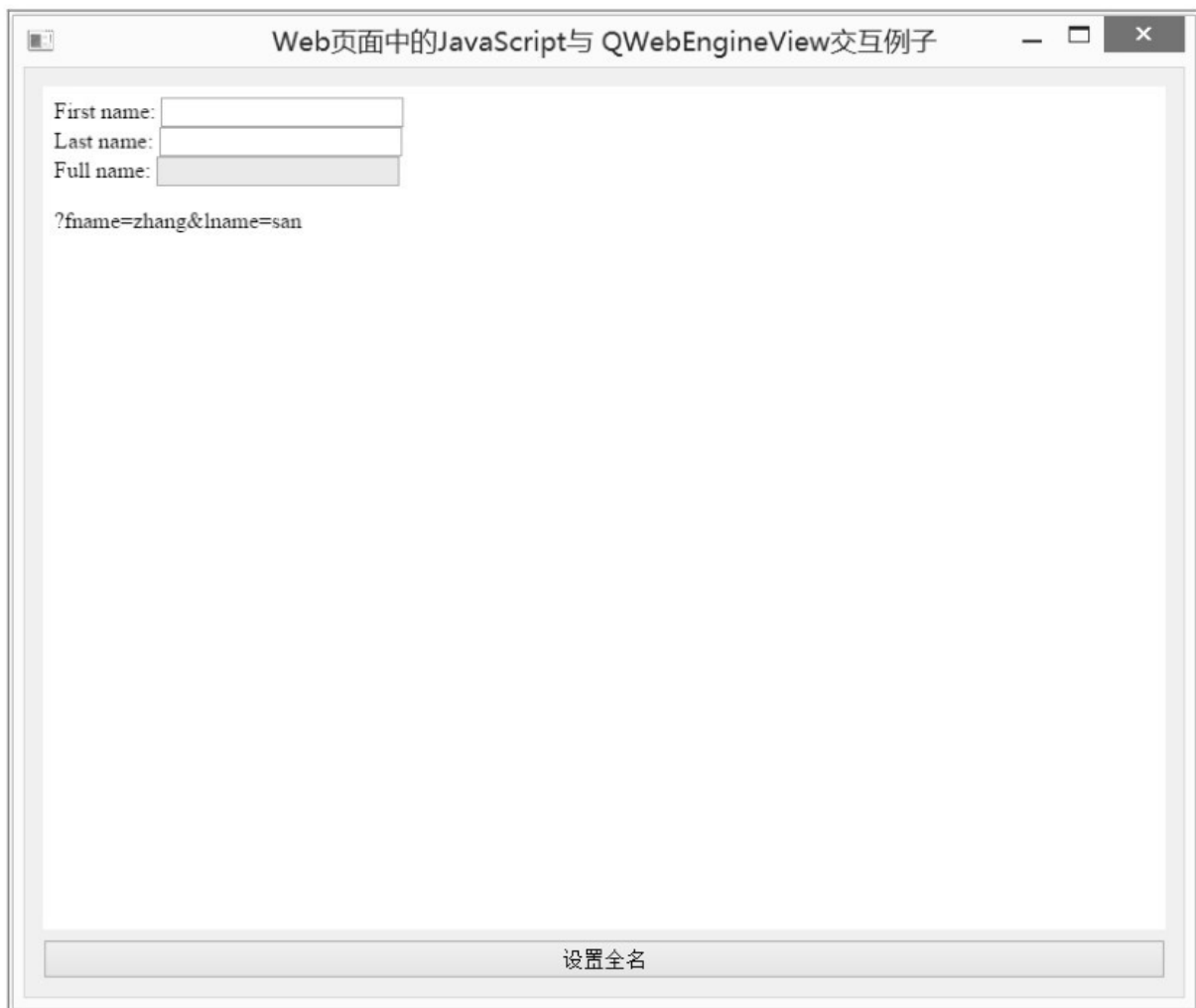


图5-48

代码如下

在main.cpp中，我们使用QWebEngineView来创建Web View。在view.cpp中，我们使用view.page()来创建QWebEnginePage。在Web页面中，我们使用QWebEnginePage的runJavaScript()方法来执行JavaScript代码。

```
def js_callback(result):
```

```
    print(result)
```

```
def complete_name():
```

```
view.page().runJavaScript('completeAndReturnName(
);',js_callback)
```

## 5-14 JavaScriptPyQt

JavaScriptPyQtPyQtWeb  
QWebEngineViewWebWeb  
JavaScriptWebJavaScript  
PyQtPyQtWeb

### 1. QWebChannel

QWebChannelWebJavaScript

```
channel=QWebChannel()
myObj=MySharedObject()
channel.registerObject( "bridge",myObj )
view.page().setWebChannel(channel)
```

### 2. PyQt

QWidgetQObject

```
class MySharedObject(QWidget):
    def __init__( self):
        super( MySharedObject,self).__init__()
    def _setStrValue( self,str ):
        print(''%s'% str )
    #
    strValue=pyqtProperty(str,fset=_setStrValue)
```

在 PyQt 中，我们使用 `pyqtProperty()` 来定义属性。  
在 PyQt 5 中，`pyqtProperty()` 是 `PyQt5.QtCore.pyqtProperty()`。  
在 Python 中，我们使用 `property()` 来定义属性。  
API 文档

`PyQt5.QtCore.pyqtProperty(type[, fget=None[, fset=None[, freset=None[, fdel=None[, doc=None[, designable=True[, scriptable=True[, stored=True[, user=False[, constant=False[, final=False[, notify=None[, revision=0]]]]]]]]]]))`  
`pyqtProperty` 5-23

5-23

参 数	说 明
type	必填，属性的类型
fget	可选，用于获取属性的值
fset	可选，用于设置属性的值
freset	可选，用于将属性的值重置为它的默认值
fdel	可选，用于删除属性

续

参 数	说 明
Doc	可选，属性的文档字符串
designable	可选，设置 Qt DESIGNABLE 标志
scriptable	可选，设置 Qt SCRIPTABLE 标志
stored	可选，设置 Qt STORED 标志
user	可选，设置 Qt USER 标志
constant	可选，设置 Qt CONSTANT 标志
final	可选，设置 Qt FINAL 标志
notify	可选，未绑定的通知信号
revision	可选，将版本导出到 QML

在 `PyqtProperty()` 中，我们使用 `setter` 和 `getter`。  
在 `PyqtProperty()` 中，我们使用 `PyQt5/Chapter05/qt05_property.py`。



index.html

Web index.html  
PyQt5/Chapter05/web/index.html

```
<html>
  <head>
    <title>A Demo Page</title>
    <meta charset="UTF-8">
    <script src="./qwebchannel.js"></script>
    <script language="javascript">
      function completeAndReturnName() {
        var fname = document.getElementById('fname').value;
        var lname = document.getElementById('lname').value;
        var full = fname + ' ' + lname;

        document.getElementById('fullname').value = full;
        document.getElementById('submit-btn').style.display =
'block';
        return full;
      }

      document.addEventListener("DOMContentLoaded", function () {

        new QWebChannel( qt.webChannelTransport, function(channel) {
          //alert('111 channel=' + channel )

          window.bridge = channel.objects.bridge;
          alert('bridge='+bridge+'\n从 pyqt 传来的参数=' +
window.bridge.strValue ) ;

        });
      });
    </script>
  </head>
  <body>
    <div>
      <input type="text" id="fname" value="First Name" />
      <input type="text" id="lname" value="Last Name" />
      <input type="button" value="Submit" id="submit-btn" />
      <div id="fullname" style="margin-top: 10px;">Full Name: <input type="text" />
    </div>
  </body>
</html>
```

```

        function onShowMsgBox() {
            //alert('window.bridge=' + window.bridge);

            if ( window.bridge) {
                var fname = document.getElementById('fname').value;
                window.bridge.strValue = fname;
            }
        }

    </script>
</head>

<body>
    <form>
        <label for="姓名">user name:</label>
        <input type="text" name="fname" id="fname"></input>
        <br />
        <input type="button" value="传递参数到 pyqt"
onclick="onShowMsgBox()" ">
        <input type="reset" value='重置' />
    </form>
</body>
</html>

```

完整代码在qwebchannel.js中，Qt中也有完整的代码，
<https://code.csdn.net/tujiaw/webengineview/tree/master/qwebchannel.js>

```

<script src="./qwebchannel.js"></script>
WebEngineView::PyQtChannel(channel.objects.bridge
bridgePyQtChannel()
document.addEventListener("DOMContentLoaded",fun
ction () {

```

```

        new QWebChannel(
qt.webChannelTransport,function(channel) {
    window.bridge=channel.objects.bridge;
});
});

```

#### 4.

PyQt5/Chapter05/qt502\_webviewJs02.py

```

from PyQt5.QtWidgets import QApplication ,QWidget
,QVBoxLayout
from PyQt5.QtWebEngineWidgets import
QWebEngineView
from PyQt5.QtCore import QUrl
from MySharedObject import MySharedObject
from PyQt5.QtWebChannel import QWebChannel
import sys
# 
app=QApplication(sys.argv)
win=QWidget()
win.setWindowTitle('Web  JavaScript 
QWebEngineView')
# 
layout=QVBoxLayout()
win.setLayout(layout)
# 
view=QWebEngineView()
htmlUrl='http://127.0.0.1:8020/web/index.html'

```

```

view.load( QUrl( htmlUrl ))
# 创建 QWebChannel 并连接到 PyQt 中的 JavaScript
channel=QWebChannel( )
myObj=MySharedObject()
channel.registerObject( "bridge",myObj )
view.page().setWebChannel(channel)
# 在 QWebEngineView 中添加 button 到布局中
layout.addWidget(view)
# 显示窗口
win.show()
sys.exit(app.exec_())

```

在 `MySharedObject` 类中实现 `PyQt` 中的 `QWidget` 接口

```

from PyQt5.QtCore import QObject
from PyQt5.QtCore import pyqtProperty
from PyQt5.QtWidgets import QWidget, QMessageBox

class MySharedObject(QWidget):

```



```

def __init__( self):
    super( MySharedObject, self).__init__()

def _getStrValue( self):
    # 设置参数
    return '100'

def _setStrValue( self, str ):
    # 获得参数
    print('获得页面参数: %s'% str )
    QMessageBox.information(self,"Information", '获得页面参数: %s'%
str )

# 需要定义对外发布的方法
strValue = pyqtProperty(str, fget=_getStrValue, fset=_setStrValue)

```

□□

□□□□MySharedObject□□□□QWidget□□□□□□□□□□PyQt□□□  
□□□□□□□□□□ Python □□□□□□□□□□□□□□□ Pyqt □□□□□□□□□  
MySharedObject□□QObject□□□□□□□  
□□□□□□□□□□5-49□□5-50□□□□



图5-49

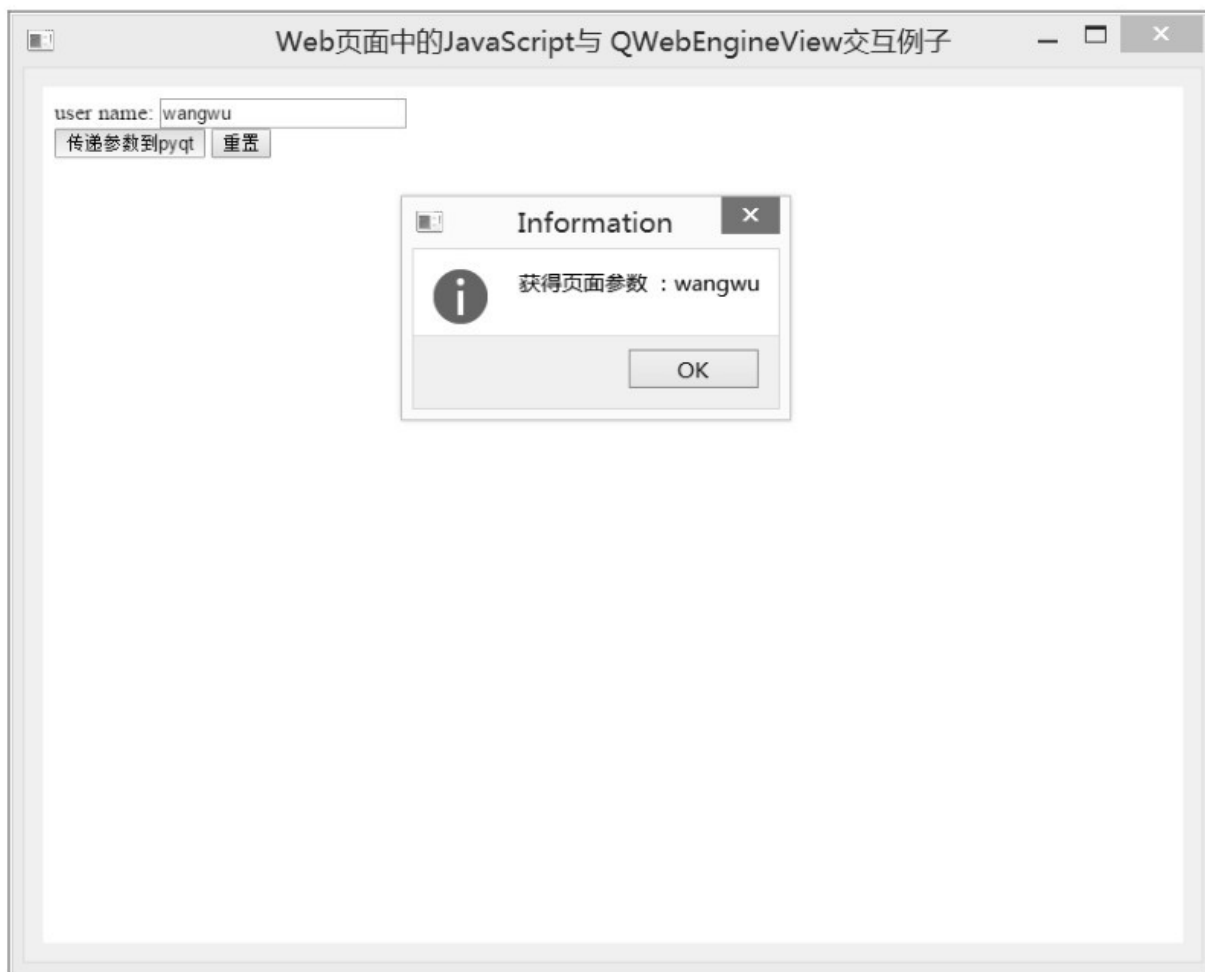


图5-50

图5-13和图5-14展示了QWebEngineView和WebPage之间的交互。QWebChannel用于在Qt和JavaScript之间传递数据。

# 第6章 PyQt 5 入门

## 6.1 用户体验

用户体验是指用户在使用产品或服务时的感受和体验。它包括用户对产品或服务的使用过程、使用结果、使用成本、使用风险、使用收益等方面的感受和体验。

用户体验的好坏直接影响到用户对产品或服务的使用意愿和忠诚度。因此，在设计产品或服务时，必须充分考虑用户体验，以提高用户满意度和忠诚度。

6-1

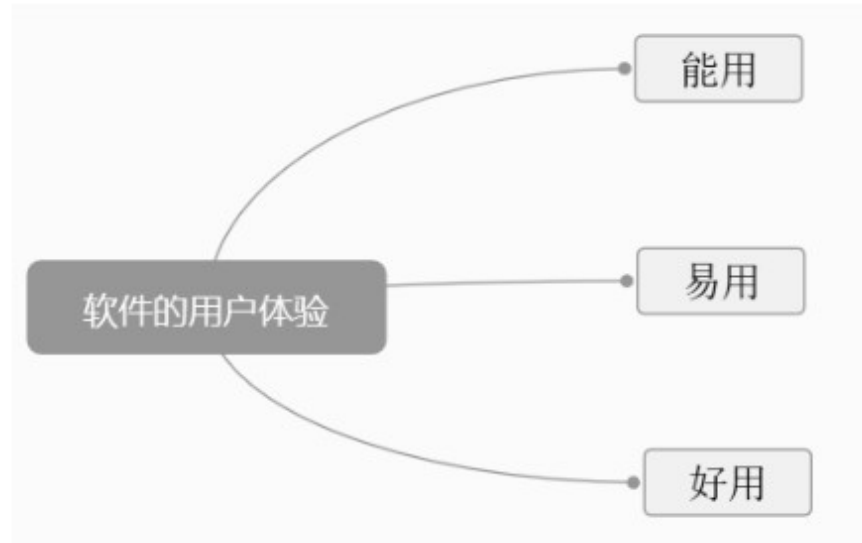


图6-1

“能用”是指用户能够使用产品或服务，即产品或服务具有基本的可用性。它包括用户能够理解产品或服务的基本功能、能够操作产品或服务、能够完成基本的任务等。





PyQt5 6-3

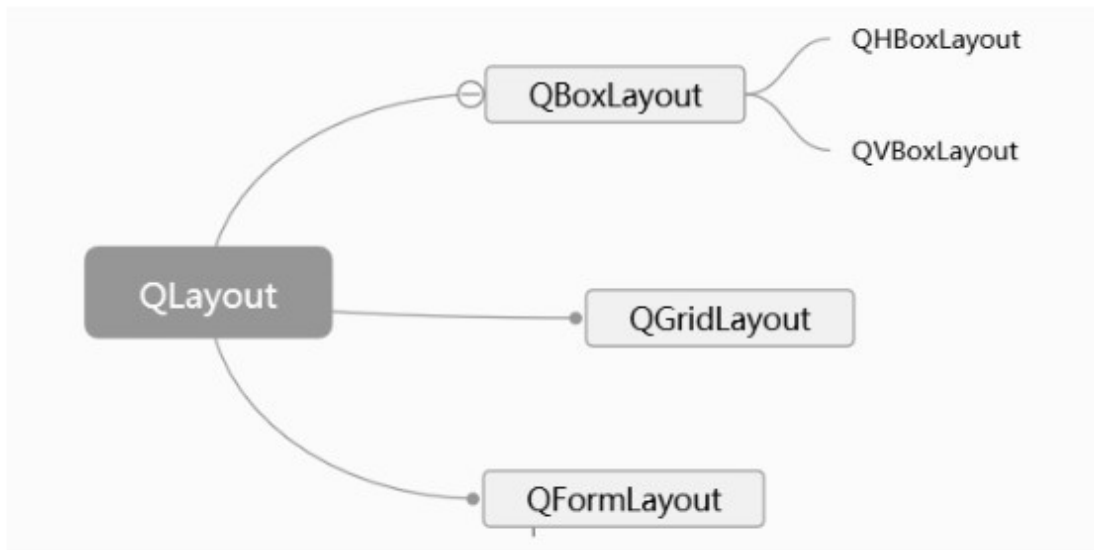


图6-3

## 6.3 PyQt 5 绝对定位

绝对定位布局 (Absolute Positioning Layout) 是一种将子部件放置在父部件中的特定位置 (x, y) 的布局管理器。子部件的位置是相对于父部件的左上角 (0,0) 来定义的。这种布局管理器适用于需要精确控制子部件位置的情况。

在 PyQt5/Chapter06/qt06\_absoPosition.py 文件中，我们使用 PyQt 5 的绝对定位布局来创建一个简单的窗口。

```

import sys
from PyQt5.QtWidgets import QWidget, QLabel, QApplication

class Example(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):

```

```

        lbl1 = QLabel('欢迎', self)
        lbl1.move(15, 10)

        lbl2 = QLabel('学习', self)
        lbl2.move(35, 40)

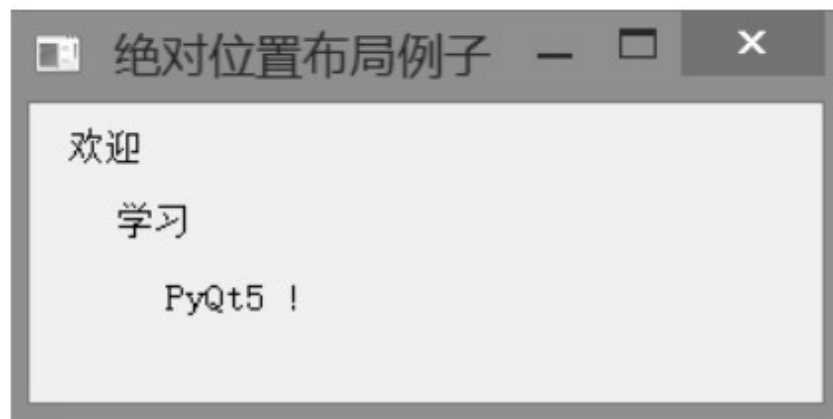
        lbl3 = QLabel('PyQt5 !', self)
        lbl3.move(55, 70)

        self.setGeometry(300, 300, 320, 120)
        self.setWindowTitle('绝对位置布局例子')

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = Example()
    demo.show()
    sys.exit(app.exec_())

```

绝对位置布局例子6-4





```

    lbl1.move(15,10)
    x=15,y=10
    lbl1=QLabel(' ',self)
    lbl1.move(15,10)
    ...
    ● ...
    ...
    ● ...
    ● ...
    ● ...
    ● ...

```

## 6.4 QBoxLayout

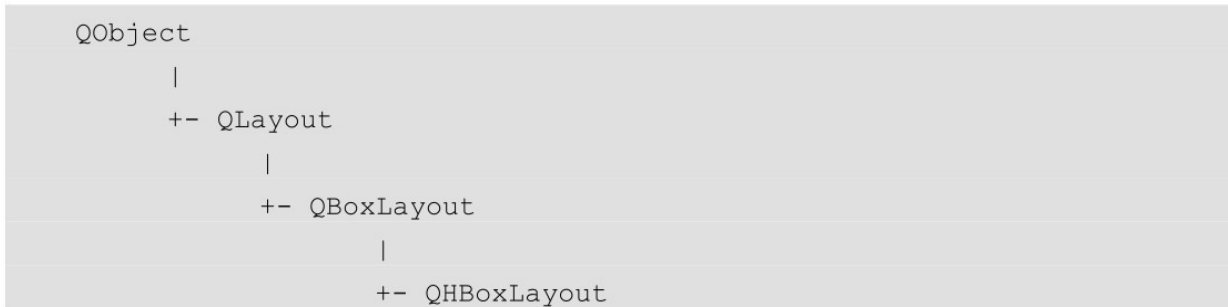
QBoxLayout 包含 QHBoxLayout 和 QVBoxLayout

### 6.4.1 QHBoxLayout

QHBoxLayout 包含 QHBoxLayout

方 法	描 述
addLayout(self, QLayout, stretch = 0)	在窗口的右边添加布局，使用 stretch（伸缩量）进行伸缩，伸缩量默认为 0
addWidget(self, QWidget, stretch, Qt.Alignment alignment)	在布局中添加控件： <ul style="list-style-type: none"><li>• stretch（伸缩量），只适用于 QVBoxLayout，控件和窗口会随着伸缩量的变大而增大</li><li>• alignment，指定对齐的方式</li></ul>
addSpacing(self, int)	设置各控件的上下间距，通过该方法可以增加额外的空间

## QHBoxLayout



## QHBoxLayout

图 6-2

参 数	描 述
Qt.AlignLeft	水平方向居左对齐
Qt.AlignRight	水平方向居右对齐
Qt.AlignCenter	水平方向居中对齐
Qt.AlignJustify	水平方向两端对齐
Qt.AlignTop	垂直方向靠上对齐
Qt.AlignBottom	垂直方向靠下对齐
Qt.AlignVCenter	垂直方向居中对齐

```
PyQt5/Chapter06/qt06_hboxLayout.py
from PyQt5.QtCore import Qt
class Winform(QWidget):
    def __init__(self,parent=None):
        super(Winform,self).__init__(parent)
        self.setWindowTitle(" ")
```

```
# 水平布局管理例子
hlayout=QHBoxLayout()
hlayout.addWidget( QPushButton(str(1)))
hlayout.addWidget( QPushButton(str(2)))
hlayout.addWidget( QPushButton(str(3)))
hlayout.addWidget( QPushButton(str(4)))
hlayout.addWidget( QPushButton(str(5)))
self.setLayout(hlayout)
```

图6-5

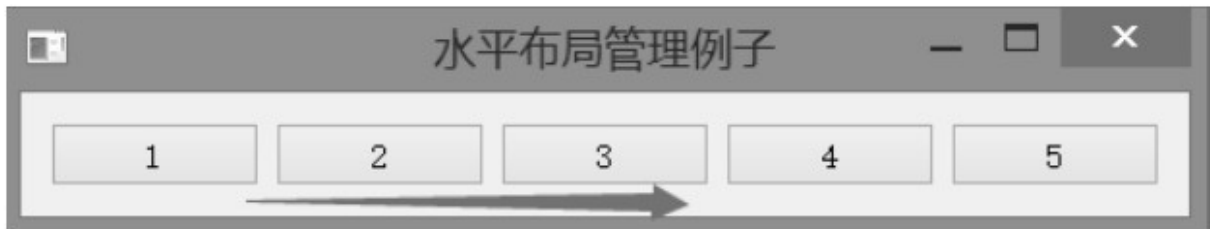


图6-5

图6-5展示了水平布局管理例子的运行结果。图中显示了五个按钮，分别标有数字1、2、3、4、5，它们按照从左到右的顺序排列。图中还显示了Qt.Alignment和PyQt5/Chapter06/qt06\_boxLayout02.py中的Qt.AlignLeft和Qt.AlignBottom等属性。

```
# 水平布局管理例子
hlayout=QHBoxLayout()
# 添加按钮
hlayout.addWidget( QPushButton(str(1)) ,0 ,|
Qt.AlignTop)
hlayout.addWidget( QPushButton(str(2)) ,0
,Qt.AlignLeft | Qt.AlignTop)
hlayout.addWidget( QPushButton(str(3)))
```

```

# 设置间距
layout.addWidget( QPushButton(str(4)) ,0
,Qt.AlignLeft |
Qt.AlignBottom )
layout.addWidget( QPushButton(str(5)),0
,Qt.AlignLeft |
Qt.AlignBottom)

```

图6-6



图6-6

在代码中，`setSpacing(int)` 用于设置布局中组件之间的间距。在 `PyQt5/Chapter06/qt06_boxLayout01.py` 文件中，`setSpacing(0)` 用于设置间距为0。

```

# 创建布局
layout=QHBoxLayout()
layout.addWidget( QPushButton(str(1)) )
layout.addWidget( QPushButton(str(2)) )
layout.addWidget( QPushButton(str(3)) )
layout.addWidget( QPushButton(str(4)) )
layout.addWidget( QPushButton(str(5)) )

```

```
# 设置间距
hlayout.setSpacing( 0 )
# 运行程序，如图6-7所示
```

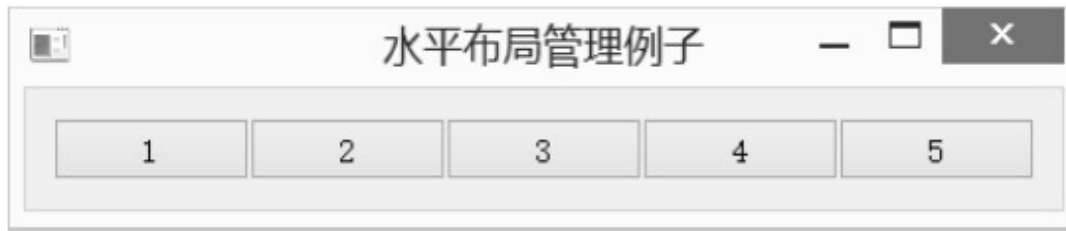


图6-7

### 6.4.2 QVBoxLayout

本节介绍 QVBoxLayout 垂直布局管理器。本节程序文件位于  
PyQt5/Chapter06/qt06\_vboxLayout.py

```
class Winform(QWidget):
    def __init__(self,parent=None):
        super(Winform,self).__init__(parent)
        self.setWindowTitle("垂直布局管理例子")
        # 创建垂直布局管理器
        vlayout=QVBoxLayout()
        vlayout.addWidget( QPushButton(str(1)))
        vlayout.addWidget( QPushButton(str(2)))
        vlayout.addWidget( QPushButton(str(3)))
        vlayout.addWidget( QPushButton(str(4)))
        vlayout.addWidget( QPushButton(str(5)))
        self.setLayout(vlayout)
```

运行程序，如图6-8所示

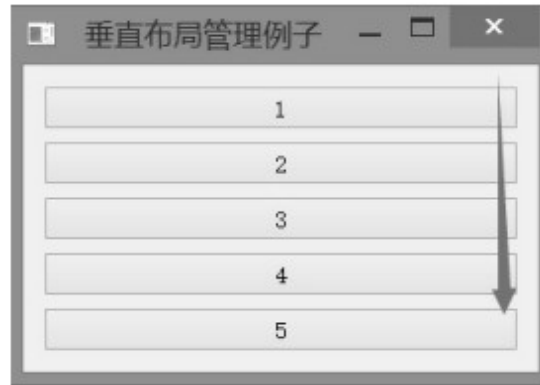


图6-8

### 6.4.3 addStretch()函数

addStretch()函数在布局管理器中增加一个可伸缩的控件（QSpaceItem），0为最小值，并且将stretch作为伸缩量添加到布局末尾  
addStretch()函数在布局管理器中增加一个可伸缩的控件（QSpaceItem），0为最小值，并且将stretch作为伸缩量添加到布局末尾

图6-3

函 数	描 述
QBoxLayout.addStretch ( int stretch = 0 )	addStretch()函数在布局管理器中增加一个可伸缩的控件（QSpaceItem），0为最小值，并且将stretch作为伸缩量添加到布局末尾 stretch 参数表示均分的比例，默认值为0

PyQt5/Chapter06/qt06\_layoutAddStretch01.py  
PyQt5/Chapter06/qt06\_layoutAddStretch01.py

```
from PyQt5.QtWidgets import QApplication ,QWidget, QVBoxLayout ,
QHBoxLayout ,QPushButton
import sys

class WindowDemo(QWidget):
    def __init__(self ):
        super().__init__()

        btn1 = QPushButton(self)
        btn2 = QPushButton(self)
```

```

btn3 = QPushButton(self)
btn1.setText('button 1')
btn2.setText('button 2')
btn3.setText('button 3')

hbox = QHBoxLayout()
# 设置伸缩量为 1
hbox.addStretch(1)
hbox.addWidget( btn1 )
# 设置伸缩量为 1
hbox.addStretch(1)
hbox.addWidget( btn2 )
# 设置伸缩量为 1
hbox.addStretch(1)
hbox.addWidget( btn3 )
# 设置伸缩量为 1
hbox.addStretch(1)

self.setLayout(hbox)
self.setWindowTitle("addStretch 例子")

if __name__ == "__main__":
    app = QApplication(sys.argv)
    win = WindowDemo()
    win.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□6-9□□□□□□□□□□6-10□□□□



□6-9



图6-10

addStretch() 添加一个弹性空间，其比例默认为1:1:1:1，即弹性空间与四个子控件的比例为1:1:1:1。

PyQt5/Chapter06/qt06\_layoutAddStretch02.py 使用 addStretch() 添加弹性空间。

```
layout=QHBoxLayout()
# 添加弹性空间
layout.addStretch(0)
layout.addWidget( QPushButton(str(1)) )
layout.addWidget( QPushButton(str(2)) )
layout.addWidget( QPushButton(str(3)) )
layout.addWidget( QPushButton(str(4)) )
layout.addWidget( QPushButton(str(5)) )
```

图6-11 运行结果



图6-11

PyQt5/Chapter06/qt06\_layoutAddStretch03.py 使用 addStretch() 添加弹性空间。

```
layout=QHBoxLayout()
layout.addWidget( QPushButton(str(1)) )
```



```

layout.addWidget( QPushButton(str(2)) )
layout.addWidget( QPushButton(str(3)) )
layout.addWidget( QPushButton(str(4)) )
layout.addWidget( QPushButton(str(5)) )
# 空行
layout.addStretch(0)

```

图6-12



图6-12

## 6.5 QGridLayout

QGridLayout 是 Qt 提供的一个二维网格布局管理器。它允许你将 widget 按照行和列的方式添加到窗口中。addWidget() 方法用于将 widget 添加到布局中，addLayout() 方法用于将子布局添加到布局中。Layout 对象可以通过 addWidget() 方法添加到主布局中。

QGridLayout 图6-4

图6-4

方 法	描 述
<code>addWidget ( QWidget widget, int row, int col, int alignment = 0 )</code>	给网格布局添加控件,设置指定的行和列。起始位置(top-left position)的默认值是(0, 0)。 <ul style="list-style-type: none"><li>• widget: 所添加的控件</li><li>• row: 控件的行数, 默认从 0 开始</li><li>• column: 控件的列数, 默认从 0 开始</li><li>• alignment: 对齐方式</li></ul>
<code>addWidget(QWidget widget, int fromRow, int fromColumn, int rowSpan, int columnSpan, Qt.Alignment alignment = 0)</code>	所添加的控件跨越很多行或者列时, 使用这个函数。 <ul style="list-style-type: none"><li>• widget: 所添加的控件</li><li>• fromRow: 控件的起始行数</li><li>• fromColumn: 控件的起始列数</li><li>• rowSpan: 控件跨越的行数</li><li>• columnSpan: 控件跨越的列数</li><li>• alignment: 对齐方式</li></ul>
<code>setSpacing ( int spacing )</code>	设置控件在水平和垂直方向的间隔

## QGridLayout



### 6.5.1

PyQt5/Chapter06/qt06\_vboxLayout01.py

```
import sys
from PyQt5.QtWidgets import QApplication, QWidget, QGridLayout,
QPushButton

class Winform(QWidget):
    def __init__(self, parent=None):
        super(Winform, self).__init__(parent)
```

```

        self.initUI()

    def initUI(self):
        # 1
        grid = QGridLayout()
        self.setLayout(grid)

        # 2
        names = ['Cls', 'Back', '', 'Close',
                 '7', '8', '9', '/',
                 '4', '5', '6', '*',
                 '1', '2', '3', '-',
                 '0', '.', '=', '+']

        # 3
        positions = [(i,j) for i in range(5) for j in range(4)]

        # 4
        for position, name in zip(positions, names):
            if name == '':
                continue

            button = QPushButton(name)
            grid.addWidget(button, *position)

        self.move(300, 150)
        self.setWindowTitle('网格布局管理例子')

if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = Winform()
    form.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□6-13□□□

□1□□□□□□QGridLayout□□□□□□□□□□□□

□2□□□□□□□□□□□□□□

□3□□□□□□□□□□□□□□□□

第4步使用addWidget()添加按钮

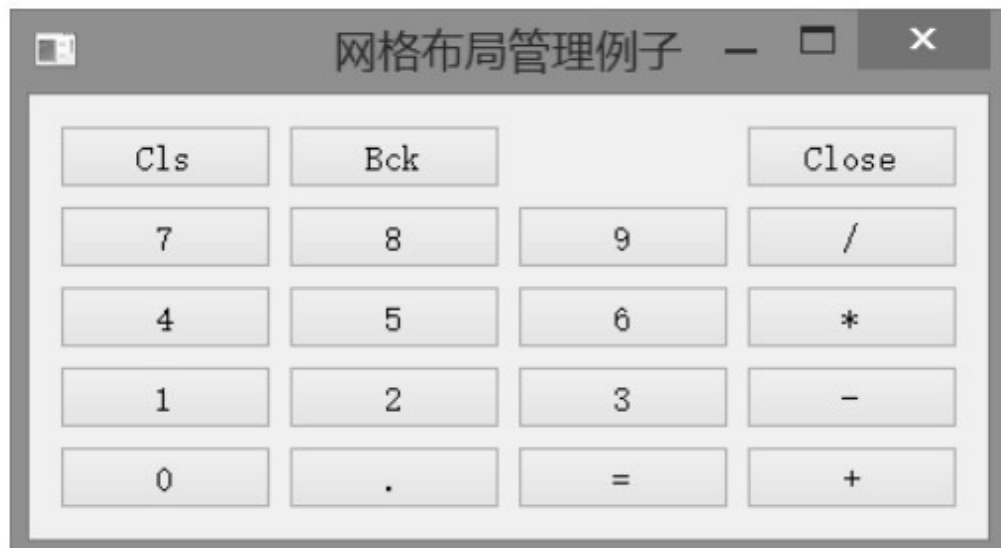


图6-13

## [6.5.2 使用vboxLayout](#)

下面使用PyQt5/Chapter06/qt06\_vboxLayout02.py实现

图

```
import sys
from PyQt5.QtWidgets import (QWidget, QLabel, QLineEdit, QTextEdit,
QGridLayout, QApplication)

class Winform(QWidget):
    def __init__(self, parent=None):
        super(Winform, self).__init__(parent)
        self.initUI()

    def initUI(self):
        title = QLabel('标题')
        author = QLabel('提交人')
        review = QLabel('申告内容')

        titleEdit = QLineEdit()
        authorEdit = QLineEdit()
        reviewEdit = QTextEdit()

        grid = QGridLayout()
        grid.setSpacing(10)

        grid.addWidget(title, 1, 0)
        grid.addWidget(titleEdit, 1, 1)

        grid.addWidget(author, 2, 0)
        grid.addWidget(authorEdit, 2, 1)

        grid.addWidget(review, 3, 0)
```

```

        grid.addWidget(reviewEdit, 3, 1, 5, 1)

        self.setLayout(grid)

        self.setGeometry(300, 300, 350, 300)
        self.setWindowTitle('故障申告')

if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = Winform()
    form.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□6-14□□□



The image shows a Qt window titled "故障申告" (Fault Report). The window has a standard title bar with a close button. Inside the window, there are three input fields: "标题" (Title), "提交人" (Submitter), and "申告内容" (Report Content). The "申告内容" field is a large text area.

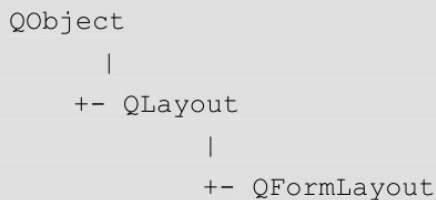
□6-14

□□□□□

```
grid.addWidget(titleLabel,1,0)
titleLabel=QGridLayout(1,0)
grid.addWidget(titleEdit,1,1)
titleEdit=QGridLayout(1,1)
grid.addWidget(contentLabel,3,0)
contentLabel=QGridLayout(3,0)
grid.addWidget(contentEdit,3,1,5,1)
contentEdit=QGridLayout(3,1,5,1)
```

## 6.6 QFormLayout

```
QFormLayout[label-field]
label field label field
QFormLayout
```



PyQt5/Chapter06/qt06\_formLayout.py

```
class Winform(QWidget):
    def __init__(self,parent=None):
        super(Winform,self).__init__(parent)
        self.setWindowTitle("□□□□□□□□")
        self.resize(400,100)
```

□□□□□□□□□□6-15□□□

[illegible]

### 6.7.1 □□□□□□□□□□



## PyQt5/Chapter06/qt06\_nestLayout01.py

```
import sys

from PyQt5.QtWidgets import QApplication ,QWidget , QHBoxLayout,
QVBoxLayout, QGridLayout , QFormLayout, QPushButton

class MyWindow( QWidget):

    def __init__(self):
        super().__init__()
        self.setWindowTitle('嵌套布局示例')

        # 全局布局（2种）：水平
        wlayout = QHBoxLayout()
        # 局部布局（4种）：水平、垂直、网格、表单
        hlayout = QHBoxLayout()
        vlayout = QVBoxLayout()
        glayout = QGridLayout()
        formlayout = QFormLayout()

        # 为局部布局添加控件（例如：按钮）
        hlayout.addWidget( QPushButton(str(1)) )
        hlayout.addWidget( QPushButton(str(2)) )
        vlayout.addWidget( QPushButton(str(3)) )
        vlayout.addWidget( QPushButton(str(4)) )
        glayout.addWidget( QPushButton(str(5)) , 0, 0 )
        glayout.addWidget( QPushButton(str(6)) , 0, 1 )
        glayout.addWidget( QPushButton(str(7)) , 1, 0 )
        glayout.addWidget( QPushButton(str(8)) , 1, 1 )
        formlayout.addWidget( QPushButton(str(9)) )
        formlayout.addWidget( QPushButton(str(10)) )
        formlayout.addWidget( QPushButton(str(11)) )
        formlayout.addWidget( QPushButton(str(12)) )
```

```

# 准备 4 个控件
hwg = QWidget()
vwg = QWidget()
gwg = QWidget()
fwg = QWidget()

# 使用 4 个控件设置局部布局
hwg.setLayout(hlayout)
vwg.setLayout(vlayout)
gwg.setLayout(glayout)
fwg.setLayout(formlayout)

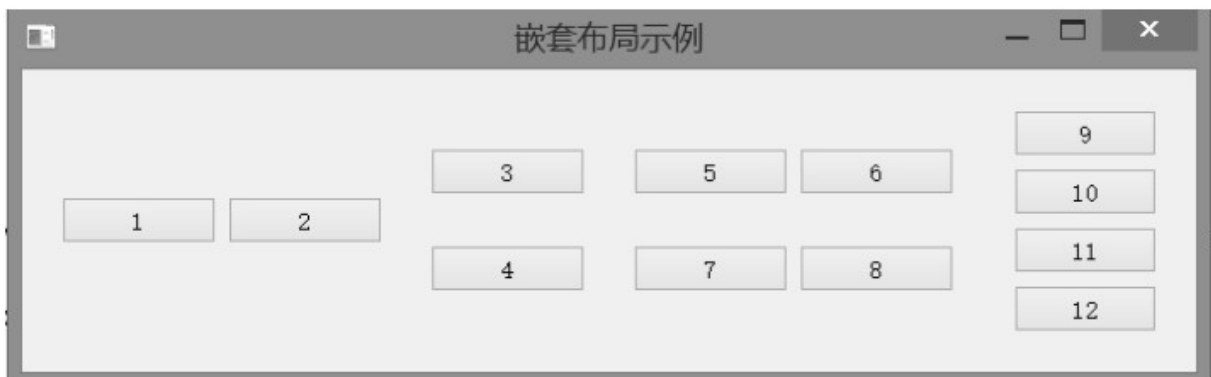
# 将 4 个控件添加到全局布局中
wlayout.addWidget(hwg)
wlayout.addWidget(vwg)
wlayout.addWidget(gwg)
wlayout.addWidget(fwg)

# 将窗口本身设置为全局布局
self.setLayout(wlayout)

if __name__=="__main__":
    app = QApplication(sys.argv)
    win = MyWindow()
    win.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□6-16□□□



□ □ □ □ □

010000

□ □ □ □ □ □ □ □ □ □ □ □

```
wlayout=QHBoxLayout()
```

□2□□□□

[illegible]

```
hlayout=QHBoxLayout()
```

```
vlayout=QVBoxLayout()
```

```
glayout=QGridLayout()
```

```
formlayout=QFormLayout()
```

#### 4 QWidget hwg vwg gwg formlayout

## hwg=QWidget()

```
vwg=QWidget()
```

```
gwg=QWidget()
```

```
fwg=QWidget()
```

## 4 QWidget

## hwg.setLayout(hlayout)

```
vwg.setLayout(vlayout)
```

```
gwg.setLayout(glayout)
```

```
fwg.setLayout(formlayout)
```

## 4 QWidget

```
wlayout.addWidget(hwg)
```

```
wlayout.addWidget(vwlg)
```

```
wlayout.addWidget(gwg)
```

```
wlayout.addWidget(fwg)
```

□ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □

```
self.setLayout(wlayout)
```

图6-17



图6-17

## 6.7.2 嵌套布局

图6-17展示了嵌套布局的示例。该窗口包含12个按钮，布局如下：

- 左侧区域包含两个按钮，分别标有数字1和2。
- 中间区域包含一个2x2的按钮网格，分别标有数字3、4、5和6。
- 右侧区域包含一个2x2的按钮网格，分别标有数字7、8、9和10。
- 最右侧区域包含一个垂直堆叠的按钮列表，分别标有数字11、12、13和14。

该布局是通过嵌套使用Qt的布局管理器实现的。具体来说，窗口首先使用一个主布局管理器，然后将各个子区域（如2x2网格和垂直堆叠）作为子布局管理器添加到主布局中。

该示例的代码文件位于PyQt5/Chapter06/qt06\_nestLayout02.py。该文件展示了如何使用Qt的布局管理器来实现嵌套布局。

```
from PyQt5.QtWidgets import *
import sys

class MyWindow(QWidget):

    def __init__(self):
        super().__init__()
        self.setWindowTitle('嵌套布局示例')
        self.resize(700, 200)

        # 全局控件（注意参数 self），用于“承载”全局布局
        wwg = QWidget(self)

        # 全局布局（注意参数 wwg）
        wl = QHBoxLayout(wwg)
        hlayout = QHBoxLayout()
        vlayout = QVBoxLayout()
        glayout = QGridLayout()
        formlayout = QFormLayout()

        # 为局部布局添加控件（例如：按钮）
        hlayout.addWidget( QPushButton(str(1)) )
        hlayout.addWidget( QPushButton(str(2)) )
        vlayout.addWidget( QPushButton(str(3)) )
        vlayout.addWidget( QPushButton(str(4)) )
        glayout.addWidget( QPushButton(str(5)) , 0, 0 )
        glayout.addWidget( QPushButton(str(6)) , 0, 1 )
        glayout.addWidget( QPushButton(str(7)) , 1, 0 )
        glayout.addWidget( QPushButton(str(8)) , 1, 1 )
        formlayout.addWidget( QPushButton(str(9)) )
        formlayout.addWidget( QPushButton(str(10)) )
        formlayout.addWidget( QPushButton(str(11)) )
        formlayout.addWidget( QPushButton(str(12)) )

        # 这里在局部布局中添加控件，然后将其添加到全局布局中
        wl.addLayout(hlayout)
```

```
wl.addLayout(vlayout)
wl.addLayout(glayout)
wl.addLayout(formlayout)

if __name__=="__main__":
    app = QApplication(sys.argv)
    win = MyWindow()
    win.show()
    sys.exit(app.exec ())
```

6-16

--	--	--	--	--

01000000000000000000000000000000

```
# self
```

```
wwg=QWidget(self)
```

□2□□□□□□□□

```
wl=QHBoxLayout(wwg)
```

□3□□□4□□□□□□

## hlayout=QHBoxLayout()

```
vlayout=QVBoxLayout()
```

```
layout=QGridLayout()
```

```
formlayout=QFormLayout()
```

4

```
layout.addWidget( QPushButton(str(1)) )
```

```
layout.addWidget( QPushButton(str(2)) )
```

```
layout.addWidget( QPushButton(str(3)) )
```

```
layout.addWidget( QPushButton(str(4)) )
```

```
layout.addWidget( QPushButton(str(5)) ,0,0 )
```

```
layout.addWidget( QPushButton(str(6)) ,0,1 )
```

```
layout.addWidget( QPushButton(str(7)) ,1,0)
```

```

layout.addWidget( QPushButton(str(8)) ,1,1)
formlayout.addWidget( QPushButton(str(9)) )
formlayout.addWidget( QPushButton(str(10)) )
formlayout.addWidget( QPushButton(str(11)) )
formlayout.addWidget( QPushButton(str(12)) )

```

54

```

wl.addLayout(hlayout)
wl.addLayout(vlayout)
wl.addLayout(glayout)
wl.addLayout(formlayout)

```

## 6.8 QSplitter

Qt 的 Layout 管理器 PyQt 提供了 QSplitter 管理器，它允许将多个小控件插入到 QSplitter 管理器中，并可以动态地调整它们的大小。

QSplitter 管理器默认是垂直方向 (Qt.Vertical)，也可以设置为水平方向 (Qt.Horizontal)。QSplitter 管理器在 6-5 中。

6-5

方 法	描 述
addWidget()	将小控件添加到 QSplitter 管理器的布局中
indexOf()	返回小控件在 QSplitter 管理器中的索引
insertWidget()	根据指定的索引将一个控件插入到 QSplitter 管理器中
setOrientation()	设置布局方向： <ul style="list-style-type: none"> <li>• Qt.Horizontal，水平方向</li> <li>• Qt.Vertical，垂直方向</li> </ul>
setSizes()	设置控件的初始化大小
count()	返回小控件在 QSplitter 管理器中的数量

## PyQt5/Chapter06/qt06\_QSplitter.py PyQt 5 QSplitter

```
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class SplitterExample(QWidget):
    def __init__(self):
        super(SplitterExample, self).__init__()
        self.initUI()

    def initUI(self):
        hbox = QHBoxLayout(self)
        self.setWindowTitle('QSplitter 例子')

        self.setGeometry(300, 300, 300, 200)
        topleft = QFrame()
        topleft.setFrameShape(QFrame.StyledPanel)
        bottom = QFrame()
        bottom.setFrameShape(QFrame.StyledPanel)
        splitter1 = QSplitter(Qt.Horizontal)
        textedit = QTextEdit()
        splitter1.addWidget(topleft)
        splitter1.addWidget(textedit)
        splitter1.setSizes([100,200])
        splitter2 = QSplitter(Qt.Vertical)
        splitter2.addWidget(splitter1)
        splitter2.addWidget(bottom)
        hbox.addWidget(splitter2)
        self.setLayout(hbox)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = SplitterExample()
    demo.show()
    sys.exit(app.exec_())
```



图6-18

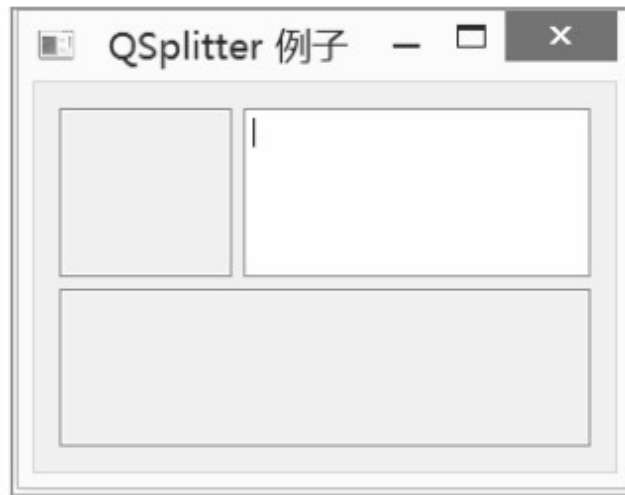


图6-18

代码

本例中，我们使用 `QSplitter` 和 `QFrame` 来创建一个窗口。  
`QSplitter` 和 `QFrame` 以及 `QTextEdit` 的初始化如下：

```
topleft=QFrame()  
topleft.setFrameShape(QFrame.StyledPanel)  
splitter1=QSplitter(Qt.Horizontal)  
textedit=QTextEdit()  
splitter1.addWidget(topleft)  
splitter1.addWidget(textedit)
```

然后，我们使用 `QSplitter` 来创建一个 `QSplitter` 窗口，并添加 `QFrame` 和 `QSplitter` 窗口。  
代码如下：

```
bottom=QFrame()  
splitter2=QSplitter(Qt.Vertical)  
splitter2.addWidget(splitter1)  
splitter2.addWidget(bottom)
```

```
hbox.addWidget(splitter2)  
self.setLayout(hbox)
```

## 第7章 PyQt 5入门

### 7.1 入门

Qt的Signal和Slot机制是Qt框架的核心，PyQt也是基于Qt框架实现的。PyQt的Qt模块提供了QObject、QWidget、QWidget等类，这些类都是QObject的子类。PyQt 5的入门教程中，通常会介绍如何使用object.signal.connect()来连接信号和槽。

PyQt的入门教程中，通常会介绍如何使用object.signal.connect()来连接信号和槽。

- 信号和槽的概念
- 信号和槽的用法
- 信号和槽的Python语法
- 信号和槽的C++语法
- 信号和槽的Qt库函数
- 信号和槽的Qt类成员函数
- 信号和槽的Qt类静态成员函数
- 信号和槽的Qt类静态成员变量

GUI是图形用户界面，Qt是跨平台的GUI框架。Qt的入门教程中，通常会介绍如何使用Qt的GUI模块来创建GUI应用程序。Qt的GUI模块提供了QWidget、QWidget等类，这些类都是QObject的子类。Qt的GUI模块还提供了许多其他的类，如QLabel、QLineEdit、QPushButton等。Qt的GUI模块还提供了许多其他的类，如QLabel、QLineEdit、QPushButton等。Qt的GUI模块还提供了许多其他的类，如QLabel、QLineEdit、QPushButton等。7-1

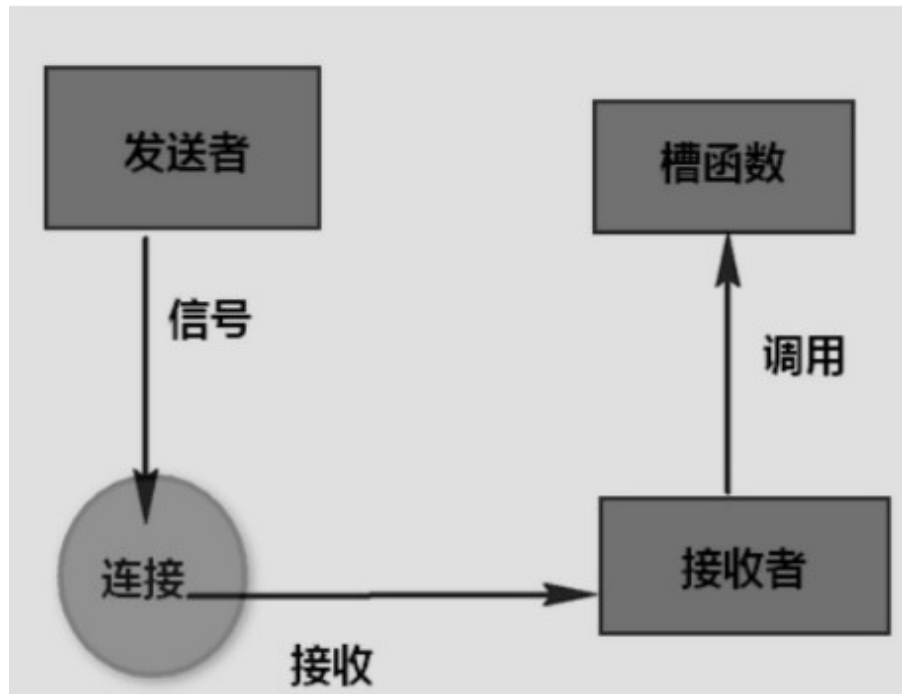


图7-1

PyQt API 文档地址：  
[http://pyqt.sourceforge.net/Docs/PyQt5/signals\\_slots.html?highlight=pyqtSignal#PyQt5.QtCore.pyqtSignal](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html?highlight=pyqtSignal#PyQt5.QtCore.pyqtSignal)

PyQt API 文档地址：  
[http://pyqt.sourceforge.net/Docs/PyQt5/signals\\_slots.html?highlight=pyqtSignal#PyQt5.QtCore.pyqtSignal](http://pyqt.sourceforge.net/Docs/PyQt5/signals_slots.html?highlight=pyqtSignal#PyQt5.QtCore.pyqtSignal)

### 7.1.1 信号槽

PyQt 提供了 `PyQt5.QtCore.pyqtSignal()` 函数，用于创建信号。该函数返回一个 `QObject` 的子类，该类具有 `pyqtSignal()` 方法，用于创建信号。该方法的返回类型是 `pyqtSignal()`。图7-2展示了信号槽的调用过程。



## 7.1.2 connect()

connect() 7-4

```
connect(slot[, type=PyQt5.QtCore.Qt.AutoConnection[, no_receiver_check=False]])
```

Connect a signal to a slot. An exception will be raised if the connection failed.

**Parameters:**

- **slot** – the slot to connect to, either a Python callable or another bound signal.
- **type** – the type of the connection to make.
- **no\_receiver\_check** – suppress the check that the underlying C++ receiver instance still exists and deliver the signal anyway.

7-4

disconnect() 7-5

```
disconnect([slot])
```

Disconnect one or more slots from a signal. An exception will be raised if the slot is not connected to the signal or if the signal has no connections at all.

**Parameters:** **slot** – the optional slot to disconnect from, either a Python callable or another bound signal. If it is omitted then all slots connected to the signal are disconnected.

7-5

emit() 7-6

```
emit(*args)
```

Emit a signal.

**Parameters:** **args** – the optional sequence of arguments to pass to any connected slots.

7-6

## 7.1.3

7.3.3 “ ”

1.





图7-8

## 2. 信号槽

在Qt中，信号槽（Signal/Slot）是用于实现对象间通信的机制。信号（Signal）是发送者发出的消息，槽（Slot）是接收者接收到的消息。通过信号槽，可以实现对象间的松耦合通信。在Python中，PyQt5提供了对信号槽的支持，可以通过`pyqtSignal`来定义信号。

PyQt5的信号槽机制与C++的Qt库类似，但在Python中，信号槽的实现更加简洁。通过`pyqtSignal`类，可以方便地定义和连接信号槽。以下是一个简单的示例，展示了如何定义和使用信号槽。

PyQt5的信号槽机制与Python的函数调用非常相似。在Python中，我们可以定义一个函数，并将其与一个信号槽连接。在PyQt5中，我们可以使用`pyqtSignal`来定义信号，并使用`connect`方法来连接信号和槽。以下是一个简单的示例，展示了如何定义和使用信号槽。

PyQt5/Chapter07/qt07\_pysignalSlot.py



```

from PyQt5.QtCore import QObject , pyqtSignal

# 信号对象
class QTypeSignal(QObject):
    # 定义一个信号
    sendmsg = pyqtSignal( object)

    def __init__( self):
        super( QTypeSignal, self).__init__()

    def run( self):
        # 发射信号
        self.sendmsg.emit('Hello Pyqt5')

# 槽对象
class QTypeSlot(QObject):
    def __init__( self):

```

```

        super( QTypeSlot, self).__init__()

    # 槽对象中的槽函数
    def get(self, msg):
        print("QSlot get msg =>" + msg)

if __name__ == '__main__':
    send = QTypeSignal()
    slot = QTypeSlot()
    # 1
    print('--- 把信号绑定到槽函数上 ---')
    send.sendmsg.connect( slot.get)
    send.run()

    # 2
    print('--- 把信号与槽函数的连接断开 ---')
    send.sendmsg.disconnect( slot.get )
    send.run()

```

```

#####
--- #####--
QSlot get msg= Hello Pyqt5
--- #####--
#####
1#####
    sendmsg=pyqtSignal(object)
2#####
    send.sendmsg.connect(slot.get)
3#####
    def get(self,msg):
        print("QSlot get msg=" + msg)
4#####
    self.sendmsg.emit('Hello Pyqt5')
5#####get()#####——'
Hello Pyqt5'#####
    send=QTypeSignal()
    slot=QTypeSlot()
    print('--- #####--')
    send.sendmsg.connect( slot.get)
    send.run()
#####get()#####
    print('--- #####--')
    send.sendmsg.disconnect( slot.get )
    send.run()
#####
#####

```

## PyQt5/Chapter07/qt07\_pysignalSlot\_2.py

```
# -*- coding: utf-8 -*-
from PyQt5.QtCore import QObject , pyqtSignal

# 信号对象
class QTypeSignal(QObject):
    # 定义一个信号
    sendmsg = pyqtSignal( str,str)

    def __init__( self):
        super( QTypeSignal, self).__init__()

    def run( self):
        # 发射信号
        self.sendmsg.emit('第一个参数','第二个参数')

# 槽对象
class QTypeSlot(QObject):
    def __init__( self):
        super( QTypeSlot, self).__init__()

    # 槽对象里的槽函数
    def get(self, msg1, msg2):
        print("QSlot get msg => " + msg1 + ' ' + msg2)

if __name__ == '__main__':
    send = QTypeSignal()
```

```
slot = QTypeSlot()

# 1
print('--- 把信号绑定到槽函数 ---')
send.sendmsg.connect( slot.get)
send.run()

# 2
print('--- 断开信号与槽函数 ---')
send.sendmsg.disconnect( slot.get )
send.run()
```

--	--	--	--	--	--	--

--- □ □ □ □ □ □ □ □ ---

QSlot get msg= 000000 000000

--- □ □ □ □ □ □ □ □ ---

### 7.3.1 “ ”

### 7.1.4 □□□□

PyQt3.4.2 安装

## 7.2 | | | | | | | | |--|--|--|--|--|--|--| | | | | | | | | |--|--|--|--|--|--|--|

### 7.2.1 □□□□□□□□

例7-9 PyQt5/Chapter07/qt07\_buildInSignalSlot01.py

```
from PyQt5.QtWidgets import *
import sys
class Winform(QWidget):
    def __init__(self,parent=None):
        super().__init__(parent)
        self.setWindowTitle('内置的信号/槽')
        self.resize(330,50 )
        btn=QPushButton('关闭',self)
        btn.clicked.connect(self.close)
if __name__=='__main__':
    app=QApplication(sys.argv)
    win=Winform()
    win.show()
    sys.exit(app.exec_())
```

例7-9 内置的信号/槽

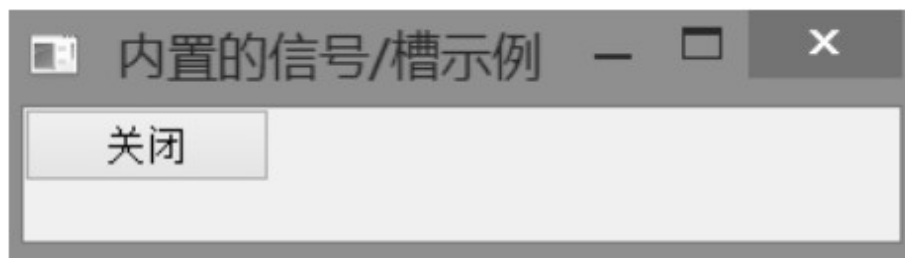


图7-9

在例7-9中，我们创建了一个名为Winform的类，它继承自QWidget。在\_\_init\_\_方法中，我们调用了self.setWindowTitle()来设置窗口标题，self.resize()来设置窗口大小，并创建了一个QPushButton按钮，将其文本设置为'关闭'。最后，我们调用了btn.clicked.connect(self.close)来将按钮的clicked信号与self.close槽函数连接起来。

## 7.2.2 信号与槽

## PyQt5/Chapter07/qt07\_buildInSignalSlot02.py

内置的信号和自定义槽函数示例

```
from PyQt5.QtWidgets import *
import sys

class Winform(QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowTitle('内置的信号和自定义槽函数示例')
        self.resize(330, 50)
        btn = QPushButton('关闭', self)
```

```
        btn.clicked.connect(self.btn_close)
```

```
    def btn_close(self):
        # 自定义槽函数
        self.close()
```

```
if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Winform()
    win.show()
    sys.exit(app.exec_())
```

7-10



图7-10

```
        clicked.connect(self.btn_close)
```

### 7.2.3 自定义信号

文件名为PyQt5/Chapter07/qt07\_buildInSignalSlot03.py

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import pyqtSignal
import sys

class Winform(QWidget):
    # 自定义信号，不带参数
    button_clicked_signal = pyqtSignal()

    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowTitle('自定义信号和内置槽函数示例')
        self.resize(330, 50)
        btn = QPushButton('关闭', self)
        # 连接信号与槽函数

        btn.clicked.connect(self.btn_clicked)
        # 接收信号，连接到槽函数
        self.button_clicked_signal.connect(self.close)

    def btn_clicked(self):
        # 发送自定义信号，无参数
        self.button_clicked_signal.emit()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Winform()
    win.show()
    sys.exit(app.exec_())
```

图7-11 运行结果



图7-11

```
button_clicked_signal.connect(self.close)
```

## 7.2.4 自定义信号

下面通过PyQt5/Chapter07/qt07\_buildInSignalSlot04.py来演示自定义信号。

```
from PyQt5.QtWidgets import *
from PyQt5.QtCore import pyqtSignal
import sys

class Winform(QWidget):
    # 自定义信号，不带参数
    button_clicked_signal = pyqtSignal()

    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowTitle('自定义信号和槽函数示例')
        self.resize(330, 50)
```



```

        btn = QPushButton('关闭', self)
        # 连接信号与槽函数
        btn.clicked.connect(self.btn_clicked)
        # 接收信号，连接到自定义的槽函数
        self.button_clicked_signal.connect(self.btn_close)

    def btn_clicked(self):
        # 发送自定义信号，无参数
        self.button_clicked_signal.emit()

    def btn_close(self):
        self.close()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Winform()
    win.show()
    sys.exit(app.exec_())

```

图7-12

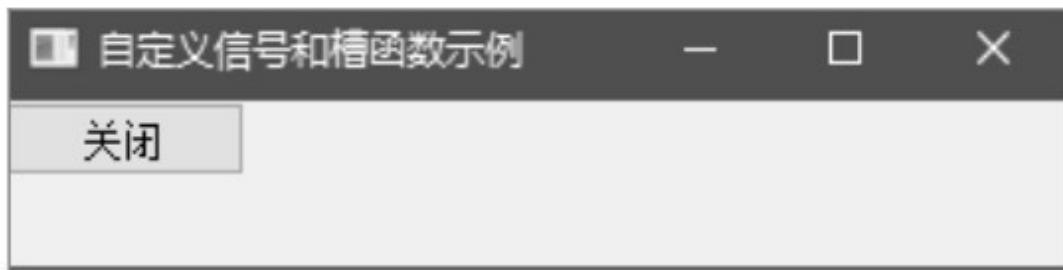


图7-12

在代码清单7-12中，我们定义了一个自定义信号 `button_clicked_signal`，并在 `btn_clicked` 槽函数中发送该信号。同时，我们在 `btn_clicked` 槽函数中调用了 `self.btn_close`，实现了点击按钮后关闭窗口的功能。

## 7.3 自定义信号和槽函数



□□□□□□□□□□□□□□□□

```
class MyWidget(QWidget):
    def setValue_NoParameters(self):
        '''无参数的槽函数'''
        pass

    def setValue_OneParameter(self, nIndex):
        '''带一个参数(整数)的槽函数'''
        pass

    def setValue_OneParameter_String(self, szIndex):
        '''带一个参数(字符串)的槽函数'''
        pass

    def setValue_TwoParameters(self, x, y):
        '''带两个参数(整数, 整数)的槽函数'''
        pass

    def setValue_TwoParameters_String(self, x, szY):
        '''带两个参数(整数, 字符串)槽函数'''
        pass
```

### 3. □□□□□□□□

□□connect□□□□□□□□□□□□□□□□

```

app = QApplication(sys.argv)
widget = MyWidget()
# 连接无参数的信号
widget.Signal_NoParameters.connect(self.setValue_NoParameters )

# 连接带一个整数参数的信号
widget.Signal_OneParameter.connect(self.setValue_OneParameter)

# 连接带一个整数参数, 经过重载的信号
widget.Signal_OneParameter_Overload[int].
    connect(self.setValue_OneParameter)

# 连接带一个整数参数, 经过重载的信号
widget.Signal_OneParameter_Overload[str].
    connect(self.setValue_OneParameter_String )

# 连接一个信号, 它有两个整数参数
widget.Signal_TwoParameters.connect(self.setValue_TwoParameters )

# 连接带两个参数(整数, 整数)的重载版本的信号
widget.Signal_TwoParameters_Overload[int,int].
    connect(self.setValue_TwoParameters )

# 连接带两个参数(整数, 字符串)的重载版本的信号
widget.Signal_TwoParameters_Overload[int,str].
    connect(self.setValue_TwoParameters_String )
widget.show()

```

## 4. 信号槽

emit 信号槽

```

class MyWidget(QWidget):
    def mousePressEvent(self,event):
        # 发出信号
        self.Signal_NoParameters.emit()
        # 发出带参数的信号
        self.Signal_OneParameter.emit(1)

```

```
# 信号槽函数(槽)的定义
self.Signal_OneParameter_Overload.emit(1)
# 信号槽函数(槽)的定义
self.Signal_OneParameter_Overload.emit("abc")
# 信号槽函数(槽,槽)的定义
self.Signal_TwoParameters.emit(1,"abc")
# 信号槽函数(槽,槽)的定义
self.Signal_TwoParameters_Overload.emit(1,2)
# 信号槽函数(槽,槽)的定义
self.Signal_TwoParameters_Overload.emit (1,"abc")
```

## 5. 运行

运行文件PyQt5/Chapter07/qt07\_signalSlot02.py

```
from PyQt5.QtCore import QObject , pyqtSignal

class CustSignal(QObject):

    #声明无参数的信号
    signal1 = pyqtSignal()

    #声明带一个 int 类型参数的信号
    signal2 = pyqtSignal(int)

    #声明带 int 和 str 类型参数的信号
    signal3 = pyqtSignal(int,str)

    #声明带一个列表类型参数的信号
    signal4 = pyqtSignal(list)

    #声明带一个字典类型参数的信号
    signal5 = pyqtSignal(dict)

    #声明一个多重载版本的信号，包括带 int 和 str 类型参数的信号和带 str 类型参数的信号
    signal6 = pyqtSignal([int,str], [str])

    def __init__(self,parent=None):
        super(CustSignal,self).__init__(parent)

    #将信号连接到指定槽函数
    self.signal1.connect(self.signalCall1)
    self.signal2.connect(self.signalCall2)
```

```
self.signal3.connect(self.signalCall3)
self.signal4.connect(self.signalCall4)
self.signal5.connect(self.signalCall5)
self.signal6[int,str].connect(self.signalCall6)
self.signal6[str].connect(self.signalCall6OverLoad)

#发射信号
self.signal1.emit()
self.signal2.emit(1)
self.signal3.emit(1,"text")
self.signal4.emit([1,2,3,4])
self.signal5.emit({"name":"wangwu","age":"25"})
self.signal6[int,str].emit(1,"text")
self.signal6[str].emit("text")

def signalCall1(self):
    print("signal1 emit")

def signalCall2(self,val):
    print("signal2 emit,value:",val)

def signalCall3(self,val,text):
    print("signal3 emit,value:",val,text)

def signalCall4(self,val):
    print("signal4 emit,value:",val)

def signalCall5(self,val):
    print("signal5 emit,value:",val)

def signalCall6(self,val,text):
    print("signal6 emit,value:",val,text)

def signalCall6OverLoad(self,val):
    print("signal6 overload emit,value:",val)

if __name__ == '__main__':
    custSignal = CustSignal()
```





```
from PyQt5.QtWidgets import QMainWindow, QPushButton , QWidget ,
QMessageBox, QApplication, QHBoxLayout
import sys
```

```
class WinForm(QMainWindow):
    def __init__(self, parent=None):
        super(WinForm, self).__init__(parent)
        button1 = QPushButton('Button 1')
        button2 = QPushButton('Button 2')

        button1.clicked.connect(lambda: self.onButtonClick(1))
        button2.clicked.connect(lambda: self.onButtonClick(2))

        layout = QHBoxLayout()
        layout.addWidget(button1)
        layout.addWidget(button2)

        main_frame = QWidget()
```

```
        main_frame.setLayout(layout)
        self.setCentralWidget(main_frame)

    def onButtonClick(self, n):
        print('Button {0} 被按下了'.format(n))
        QMessageBox.information(self, "信息提示框", 'Button {0}
clicked'.format(n))

if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = WinForm()
    form.show()
    sys.exit(app.exec_())
```

□□□□□□□□□□7-13□□7-14□□□



图7-13

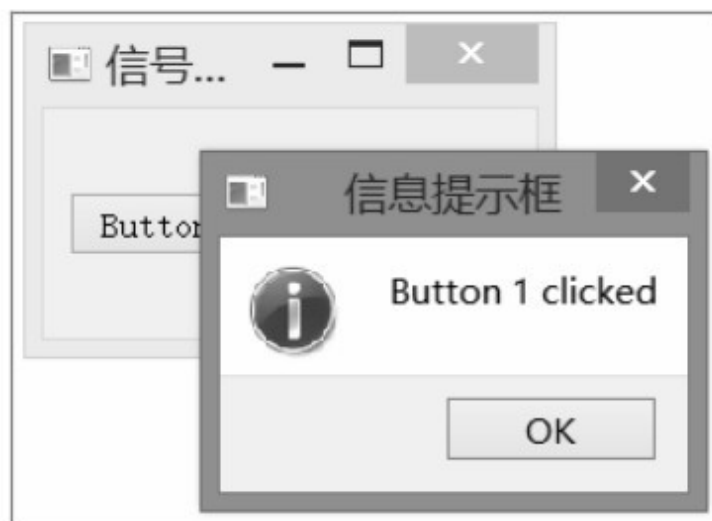


图7-14

下面代码  
 为“Button 1”添加单击事件槽函数，槽函数为“Button 1 clicked”  
 Python代码如下  
 Button 1 单击  
 槽函数 onButtonClick() 使用 lambda 表达式  
 实现单击事件槽函数，槽函数调用 QMessageBox 的 info 方法显示信息提示框  
 代码如下  
 使用 functools 模块的 partial 函数  
 PyQt5/Chapter07/qt07\_winSignalSlot05.py 代码如下

```

button1.clicked.connect(partial(self.onButtonClick,1))
button2.clicked.connect(partial(self.onButtonClick,2))
#####lambda#####

```

### 7.3.3 连接槽

```

#####
@PyQt5.QtCore.pyqtSlot()
def on_#####(self,):
    pass
#####
QMetaObject.connectSlotsByName(QObject)
#####“#####”#####setObjectName#####
#####on + #####setObjectName##### +#####
#####
#####PyQt5/Chapter07/qt07_connSlotsByName.py#####
#####

```

```

from PyQt5 import QtCore
from PyQt5.QtWidgets import QApplication ,QWidget ,QHBoxLayout ,
QPushButton
import sys

class CustWidget( QWidget ):

    def __init__(self, parent=None):
        super(CustWidget, self).__init__(parent)

        self.okButton = QPushButton("OK", self)
        #使用 setObjectName 设置对象名称
        self.okButton.setObjectName("okButton")
        layout = QHBoxLayout()
        layout.addWidget(self.okButton)
        self.setLayout(layout)
        QtCore.QMetaObject.connectSlotsByName(self)

    @QtCore.pyqtSlot()
    def on_okButton_clicked(self):
        print( "单击了 OK 按钮")

if __name__ == "__main__":
    app = QApplication(sys.argv)
    win = CustWidget()

    win.show()
    app.exec_()

```

□□□□□□□□□□ 7-15 □□□□“OK”□□□□□□□□□□□□□□□□



图7-15

```

from PyQt5.QtCore import QObject, Qt
from PyQt5.QtWidgets import QPushButton

class Clickable(QObject):
    """A QObject that has a clicked signal and a slot to handle it.
    It is a simple example of how to use Qt's signal-slot mechanism.
    """
    clicked = Qt.Signal()

    def __init__(self, parent=None):
        self.okButton.clicked.connect(self.okButton_clicked)

    def okButton_clicked(self):
        print("Clicked OK button")

    @QtCore.pyqtSlot()
    def on_okButton_clicked(self):
        print("Clicked OK button")

# Create an instance of Clickable
clickable = Clickable()

# Connect the clicked signal to the on_okButton_clicked slot
clickable.clicked.connect(clickable.on_okButton_clicked)

# Click the button
clickable.clicked.emit()

```

## PyQt5/Chapter07/qt07\_connSlotsByName\_2.py

```
# -*- coding: utf-8 -*-
```

```
"""
```

```
    【简介】
```

```
    信号与槽的自动连接例子
```

```
"""
```

```
from PyQt5 import QtCore
```

```
from PyQt5.QtWidgets import QApplication ,QWidget ,QHBoxLayout ,  
QPushButton
```

```
import sys
```

```
class CustWidget( QWidget ):
```

```
    def __init__(self, parent=None):
```

```
        super(CustWidget, self).__init__(parent)
```

```
        self.okButton = QPushButton("OK", self)
```

```
        #使用 setObjectName 设置对象名称
```

```
        self.okButton.setObjectName("okButton")
```

```
        layout = QHBoxLayout()
```

```
        layout.addWidget(self.okButton)
```

```
        self.setLayout(layout)
```

```
        QtCore.QMetaObject.connectSlotsByName(self)
```

```
        self.okButton.clicked.connect(self.okButton_clicked)
```

```
    def okButton_clicked(self):
```

```
        print( "单击了 OK 按钮")
```

```
if __name__ == "__main__":
```

```
    app = QApplication(sys.argv)
```

```
    win = CustWidget()
```

```
    win.show()
```

```
    sys.exit(app.exec_())
```

7-15

### 7.3.4 矩陣的乘法

[illegible]

PyQt5/Chapter07/qt07\_signalSlot03.py

```
from PyQt5.QtCore import QObject , pyqtSignal

class SignalClass(QObject):

    # 声明无参数的信号
    signal1 = pyqtSignal()

    # 声明带一个 int 类型参数的信号
    signal2 = pyqtSignal(int)

    def __init__(self, parent=None):
        super(SignalClass, self).__init__(parent)

        # 将信号 signal1 连接到 sin1Call 和 sin2Call 这两个槽函数
        self.signal1.connect(self.sin1Call)
        self.signal1.connect(self.sin2Call)

        # 将信号 signal2 连接到信号 signal1
        self.signal2.connect(self.signal1)

        # 发射信号
        self.signal1.emit()
        self.signal2.emit(1)

        # 断开 signal1、signal2 信号与各槽函数的连接
        self.signal1.disconnect(self.sin1Call)
        self.signal1.disconnect(self.sin2Call)
        self.signal2.disconnect(self.signal1)

        # 将信号 signal1 和 signal2 连接到同一个槽函数 sin1Call
        self.signal1.connect(self.sin1Call)
        self.signal2.connect(self.sin1Call)

        # 再次发射信号
        self.signal1.emit()
        self.signal2.emit(1)

    def sin1Call(self):
        print("signal-1 emit")

    def sin2Call(self):
        print("signal-2 emit")
```



```
if __name__ == '__main__':  
    signal = SignalClass()
```

□ □ □ □ □ □ □

```
signal-1 emit
signal-2 emit
signal-1 emit
signal-2 emit
signal-1 emit
signal-1 emit
```

### 7.3.5 Qt Designer

```

Qt
Designer xxx.ui xxx.py
Call xxx.py

```

```

Qt Designer
Qt Designer
Qt Designer

```

[illegible]

# F1 helpMessage

Qt Designer “Widget”  
MainWinSignalSlog02.ui Widget Box  
7-16

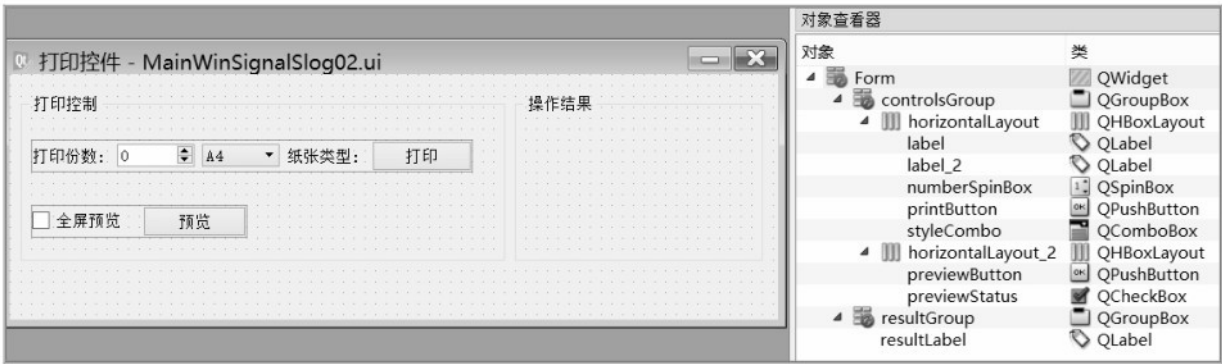


图7-16

图7-1

图7-1

控件类型	控件名称	作用
QSpinBox	numberSpinBox	显示打印的份数
QComboBox	styleCombo	显示打印的纸张类型。纸张类型包括 A3、A4 和 A5 纸
QPushButton	printButton	连接 emitPrintSignal 函数的绑定。触发自定义信号 printSignal 的发射
QCheckBox	previewStatus	是否全屏预览
QPushButton	previewButton	连接 emitPreviewSignal 函数的绑定。触发自定义信号 previewSignal 的发射
QLabel	resultLabel	显示执行结果

使用 Python 编写 MainWinSignalSlog02.ui  
 的 Python 代码文件 MainWinSignalSlog02.py 和  
 MainWinSignalSlog02.ui 的 Python 代码文件  
 pyuic5-o MainWinSignalSlog02.py  
 MainWinSignalSlog02.ui  
 的 Python 代码文件 MainWinSignalSlog02.py

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(715, 225)
        self.controlsGroup = QtWidgets.QGroupBox(Form)
        self.controlsGroup.setGeometry(QtCore.QRect(10, 20, 451, 151))
        self.controlsGroup.setObjectName("controlsGroup")
        self.widget = QtWidgets.QWidget(self.controlsGroup)
        self.widget.setGeometry(QtCore.QRect(10, 40, 411, 30))
        self.widget.setObjectName("widget")
        self.horizontalLayout = QtWidgets.QHBoxLayout(self.widget)
        self.horizontalLayout.setContentsMargins(0, 0, 0, 0)
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.label = QtWidgets.QLabel(self.widget)
        self.label.setObjectName("label")
        self.horizontalLayout.addWidget(self.label)
        self.numberSpinBox = QtWidgets.QSpinBox(self.widget)
        self.numberSpinBox.setObjectName("numberSpinBox")
        self.horizontalLayout.addWidget(self.numberSpinBox)
        self.styleCombo = QtWidgets.QComboBox(self.widget)
```

```

self.styleCombo.setObjectName("styleCombo")
self.styleCombo.addItem("")
self.styleCombo.addItem("")
self.styleCombo.addItem("")
self.horizontalLayout.addWidget(self.styleCombo)
self.label_2 = QtWidgets.QLabel(self.widget)
self.label_2.setObjectName("label_2")
self.horizontalLayout.addWidget(self.label_2)
self.printButton = QtWidgets.QPushButton(self.widget)
self.printButton.setObjectName("printButton")
self.horizontalLayout.addWidget(self.printButton)
self.widget1 = QtWidgets.QWidget(self.controlsGroup)
self.widget1.setGeometry(QtCore.QRect(10, 100, 201, 30))
self.widget1.setObjectName("widget1")
self.horizontalLayout_2 = QtWidgets.QHBoxLayout(self.widget1)
self.horizontalLayout_2.setContentsMargins(0, 0, 0, 0)
self.horizontalLayout_2.setObjectName("horizontalLayout_2")
self.previewStatus = QtWidgets.QCheckBox(self.widget1)
self.previewStatus.setObjectName("previewStatus")
self.horizontalLayout_2.addWidget(self.previewStatus)
self.previewButton = QtWidgets.QPushButton(self.widget1)
self.previewButton.setObjectName("previewButton")
self.horizontalLayout_2.addWidget(self.previewButton)
self.resultGroup = QtWidgets.QGroupBox(Form)
self.resultGroup.setGeometry(QtCore.QRect(470, 20, 231, 151))
self.resultGroup.setObjectName("resultGroup")
self.resultLabel = QtWidgets.QLabel(self.resultGroup)
self.resultLabel.setGeometry(QtCore.QRect(20, 30, 191, 101))
self.resultLabel.setObjectName("resultLabel")

self.retranslateUi(Form)
QtCore.QMetaObject.connectSlotsByName(Form)

def retranslateUi(self, Form):
    _translate = QtCore.QCoreApplication.translate
    Form.setWindowTitle(_translate("Form", "打印控件"))
    self.controlsGroup.setTitle(_translate("Form", "打印控制"))
    self.label.setText(_translate("Form", "打印份数:"))
    self.styleCombo.setItemText(0, _translate("Form", "A3"))
    self.styleCombo.setItemText(1, _translate("Form", "A4"))
    self.styleCombo.setItemText(2, _translate("Form", "A5"))
    self.label_2.setText(_translate("Form", "纸张类型:"))

```

```
self.printButton.setText(_translate("Form", "打印"))
self.previewStatus.setText(_translate("Form", "全屏预览"))
self.previewButton.setText(_translate("Form", "预览"))
self.resultGroup.setTitle(_translate("Form", "操作结果"))
self.resultLabel.setText(_translate("Form",
"<html><head/><body><p><br/></p></body></html>"))
```

CallMainWinSignalSlog02.py

```

import sys
from PyQt5.QtWidgets import QApplication , QMainWindow
from MainWinSignalSlog02 import Ui_Form
from PyQt5.QtCore import pyqtSignal , Qt

class MyMainWindow(QMainWindow, Ui_Form):
    helpSignal = pyqtSignal(str)
    printSignal = pyqtSignal(list)
    # 声明一个多重载版本的信号，包括一个带 int 和 str 类型参数的信号，以及带 str 类型参数的信号
    previewSignal = pyqtSignal([int,str],[str])

    def __init__(self, parent=None):
        super(MyMainWindow, self).__init__(parent)
        self.setupUi(self)
        self.initUI()

    def initUI(self):
        self.helpSignal.connect(self.showHelpMessage)
        self.printSignal.connect(self.printPaper)
        self.previewSignal[str].connect(self.previewPaper)
        self.previewSignal[int,str].connect(self.
previewPaperWithArgs)

        self.printButton.clicked.connect(self.emitPrintSignal)
        self.previewButton.clicked.connect(self.emitPreviewSignal)

    # 发射预览信号
    def emitPreviewSignal(self):
        if self.previewStatus.isChecked() == True:
            self.previewSignal[int,str].emit(1080," Full Screen")
        elif self.previewStatus.isChecked() == False:
            self.previewSignal[str].emit("Preview")

```

```

# 发射打印信号
def emitPrintSignal(self):
    pList = []
    pList.append(self.numberSpinBox.value() )
    pList.append(self.styleCombo.currentText())
    self.printSignal.emit(pList)

def printPaper(self, list):
    self.resultLabel.setText("打印: "+"份数: "+ str(list[0]) +" 纸张: "+str(list[1]))

def previewPaperWithArgs(self, style, text):
    self.resultLabel.setText(str(style)+text)

def previewPaper(self, text):
    self.resultLabel.setText(text)

# 重载按键事件
def keyPressEvent(self, event):
    if event.key() == Qt.Key_F1:
        self.helpSignal.emit("help message")

# 显示帮助信息
def showHelpMessage(self, message):
    self.resultLabel.setText(message)
    self.statusBar().showMessage(message)

if __name__=="__main__":
    app = QApplication(sys.argv)
    win = MyMainWindow()
    win.show()
    sys.exit(app.exec_())

```

0000000000007-17000



图7-17

代码如下

```

class PrintControl(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('打印控件')
        self.printCopies = 6
        self.paperType = 'A5'
        self.previewFlag = True
        self.helpSignal = pyqtSignal(str)
        self.printSignal = pyqtSignal(list)
        self.previewSignal = pyqtSignal([int, str], [str])
        self.previewSignal.connect(self.previewPaper)
        self.previewSignal[int, str].connect(self.previewPaperWithArgs)
        self.helpSignal.connect(self.showHelpMessage)
        self.printSignal.connect(self.printPaper)
        self.previewSignal[str].connect(self.previewPaper)
        self.previewSignal[int, str].connect(self.previewPaperWithArgs)

    def showHelpMessage(self, str):
        QMessageBox.warning(self, '帮助信息', str)

    def printPaper(self, list):
        QMessageBox.warning(self, '打印结果', '打印: 份数: %d 纸张: %s' % (list[0], list[1]))

    def previewPaper(self, str):
        QMessageBox.warning(self, '预览结果', str)

    def previewPaperWithArgs(self, args):
        QMessageBox.warning(self, '预览结果', '预览: 份数: %d 纸张: %s' % (args[0], args[1]))

```



```
#####Qt#####

```

```
def emitPreviewSignal(self):
    if self.previewStatus.isChecked()==True:
        self.previewSignal[int,str].emit(1080," Full Screen")
    elif self.previewStatus.isChecked()==False:
        self.previewSignal[str].emit("Preview")
```

```
#####Python#####printSignal#####list#####pList
```

```
def emitPrintSignal(self):
    pList=[]
    pList.append(self.numberSpinBox.value() )
    pList.append(self.styleCombo.currentText())
    self.printSignal.emit(pList)
```

```
##### keyPressEvent()##### F1 ##### Windows #####
#####F1#####
keyPressEvent()#####
```

```
# #####
def keyPressEvent(self,event):
    if event.key()==Qt.Key_F1:
        self.helpSignal.emit("help message")
```

```
#####
1#####__init__()#####
2#####str#####int#####list#####object#####float#####tuple#####dict#####
#####
3##### signal##### slot##### signal##### slot#####slot#####
#####
```

### 7.3.6 多线程信号槽

在本节中，我们将使用 `QThread` 类来创建多线程。在本节中，我们将使用 `PyQt5/Chapter07/qt07_signalSlot04.py` 来创建多线程。在本节中，我们将使用 `QThread` 类来创建多线程。

```
from PyQt5.QtWidgets import QApplication ,QWidget
from PyQt5.QtCore import QThread , pyqtSignal
import sys

class Main(QWidget):
    def __init__(self, parent = None):
        super(Main,self).__init__(parent)

        # 创建一个线程实例并设置名称、变量、信号与槽
        self.thread = MyThread()
        self.thread.setIdentity("thread1")
        self.thread.sinOut.connect(self.outText)
        self.thread.setVal(6)

    def outText(self,text):
        print(text)

class MyThread(QThread):
    sinOut = pyqtSignal(str)

    def __init__(self,parent=None):
```



BackendThread 的 update\_date 方法在 BackendThread 的 update\_date 方法中

BackendThread 的 update\_date 方法调用了 handleDisplay() 方法，该方法调用了 QLineEdit 的 setText() 方法

PyQt5/Chapter07/qt07\_signalSlotThreaad.py 文件

```

from PyQt5.QtCore import QThread , pyqtSignal, QDateTime
from PyQt5.QtWidgets import QApplication, QDialog, QLineEdit
import time
import sys

class BackendThread(QThread):
    # 通过类成员对象定义信号
    update_date = pyqtSignal(str)

    # 处理业务逻辑
    def run(self):
        while True:
            data = QDateTime.currentDateTime()
            currTime = data.toString("yyyy-MM-dd hh:mm:ss")
            self.update_date.emit( str(currTime) )
            time.sleep(1)

class Window(QDialog):
    def __init__(self):
        QDialog.__init__(self)
        self.setWindowTitle('PyQt 5 界面实时更新例子')
        self.resize(400, 100)
        self.input = QLineEdit(self)
        self.input.resize(400, 100)
        self.initUI()

    def initUI(self):
        # 创建线程
        self.backend = BackendThread()
        # 连接信号
        self.backend.update_date.connect(self.handleDisplay)
        # 开始线程
        self.backend.start()

        # 将当前时间输出到文本框
        def handleDisplay(self, data):
            self.input.setText(data)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    win = Window()
    win.show()
    sys.exit(app.exec_())

```

图7-18 图例



图7-18

## 7.4 多线程

5.3.3 PyQt 多线程  
多线程是指一个程序同时执行多个任务的能力。在 PyQt 中，多线程可以通过 `QThread` 类来实现。每个 `QThread` 对象都有一个 `run()` 方法，该方法会在该线程中执行。通过 `QThread` 的 `start()` 方法，可以启动一个新的线程。

在 PyQt 中，多线程可以通过 `QThread` 类来实现。每个 `QThread` 对象都有一个 `run()` 方法，该方法会在该线程中执行。通过 `QThread` 的 `start()` 方法，可以启动一个新的线程。

### 7.4.1 多线程

多线程是指一个程序同时执行多个任务的能力。在 PyQt 中，多线程可以通过 `QThread` 类来实现。每个 `QThread` 对象都有一个 `run()` 方法，该方法会在该线程中执行。通过 `QThread` 的 `start()` 方法，可以启动一个新的线程。

### 7.4.2 多线程



installEventFilter() 方法可以安装事件过滤器，安装事件过滤器时，需要传入一个QObject类型的对象，该对象将接收所有事件。

4. QApplication 类  
QApplication 类是 Qt 中用于管理应用程序的类，它继承自 QObject 类。QApplication 类提供了许多用于管理应用程序的方法，例如：设置应用程序名称、设置应用程序图标、设置应用程序的窗口标题等。

5. QApplication 的 notify() 方法  
PyQt 的 notify() 方法用于通知 QApplication 类，以便它可以处理事件。QApplication 的 notify() 方法将事件传递给 QApplication 类，以便它可以处理事件。

#### 7.4.4 事件过滤器

CallMainWinSignalSlog02.py 文件 keyPressEvent 方法中，我们使用 event.key() 方法获取事件的键值。如果键值是 Qt.Key\_F1，则我们调用 self.helpSignal.emit("help message") 方法，以发出帮助信号。

```
# 事件过滤器
def keyPressEvent(self, event):
    if event.key() == Qt.Key_F1:
        self.helpSignal.emit("help message")
```

PyQt5/Chapter07/event.py 文件

GUI\_Rapid GUI Programming with Python and Qt 10

PyQt 4 和 PyQt 5 的兼容性

PyQt 4 和 PyQt 5 的兼容性

PyQt 4 和 PyQt 5 的兼容性



self.text = message  
self.paintEvent()  
self.update()

update() 函数会触发 paintEvent()  
paintEvent() 函数是 QWidget 类的一个函数，update() 函数会触发 paintEvent()

```
import sys
from PyQt5.QtCore import (QEvent, QTimer, Qt)
from PyQt5.QtWidgets import (QApplication, QMenu, QWidget)
from PyQt5.QtGui import QPainter

class Widget(QWidget):
    def __init__(self, parent=None):
        super(Widget, self).__init__(parent)
        self.justDoubleClicked = False
        self.key = ""
        self.text = ""
        self.message = ""
        self.resize(400, 300)
        self.move(100, 100)
        self.setWindowTitle("Events")
        QTimer.singleShot(0, self.giveHelp) # 避免受窗口大小重绘事件的影响，
        # 可以把参数 0 改成 3000 (3 秒)，然后再运行，就可以明白这行代码的意思

    def giveHelp(self):
        self.text = "请点击这里触发追踪鼠标功能"
        self.update() # 重绘事件，也就是触发 paintEvent 函数
```

7-19



图7-19

该窗口接收来自消息队列的消息，包括 `message` 消息和 `paintEvent` 消息。图7-20和图7-21分别

```
'''重新实现关闭事件'''
def closeEvent(self, event):
    print("Closed")

'''重新实现上下文菜单事件'''
def contextMenuEvent(self, event):
    menu = QMenu(self)
    oneAction = menu.addAction("&One")
    twoAction = menu.addAction("&Two")
    oneAction.triggered.connect(self.one)
    twoAction.triggered.connect(self.two)
    if not self.message:
        menu.addSeparator()
        threeAction = menu.addAction("&Three")
        threeAction.triggered.connect(self.three)
    menu.exec_(event.globalPos())

'''上下文菜单槽函数'''
def one(self):
    self.message = "Menu option One"
```

```
        self.update()

def two(self):
    self.message = "Menu option Two"
    self.update()

def three(self):
    self.message = "Menu option Three"
    self.update()
```



图7-20



图7-21

```

    text< message<
text< message<5<

```

```

'''重新实现绘制事件'''
def paintEvent(self, event):
    text = self.text
    i = text.find("\n\n")
    if i >= 0:
        text = text[0:i]
    if self.key: # 若触发了键盘按键,则在信息文本中记录这个按键信息
        text += "\n\n你按下了: {0}".format(self.key)
    painter = QPainter(self)
    painter.setRenderHint(QPainter.TextAntialiasing)
    painter.drawText(self.rect(), Qt.AlignCenter, text) # 绘制信息文本的内容
    if self.message: # 若信息文本存在,则在底部居中绘制信息,5秒后清空信息文本并重绘
        painter.drawText(self.rect(), Qt.AlignBottom | Qt.AlignHCenter, self.message)
        QTimer.singleShot(5000, self.clearMessage)
        QTimer.singleShot(5000, self.update)

'''清空信息文本的槽函数'''
def clearMessage(self):
    self.message = ""

```

□□□□□□□□□□□□□□□□□□□□7-22□□□

""□□□□□□□□□□□□□□□□□□□□""

```

def resizeEvent(self,event):
    self.text="□□□□□□□□QSize({0},{1})".format(
        event.size().width(),event.size().height())
    self.update()

```





图7-23





图7-24



□7-25

7-26 7-27

```

'''重新实现鼠标移动事件'''
def mouseMoveEvent(self, event):
    if not self.justDoubleClicked:
        globalPos = self.mapToGlobal(event.pos())# 将窗口坐标转换为屏幕坐标
        self.text = """鼠标位置:
        窗口坐标为: QPoint({0}, {1})
        屏幕坐标为: QPoint({2}, {3})""".format(event.pos().x(),
event.pos().y(), globalPos.x(), globalPos.y())
        self.update()

'''重新实现鼠标双击事件'''
def mouseDoubleClickEvent(self, event):
    self.justDoubleClicked = True
    self.text = "你双击了鼠标"
    self.update()

```



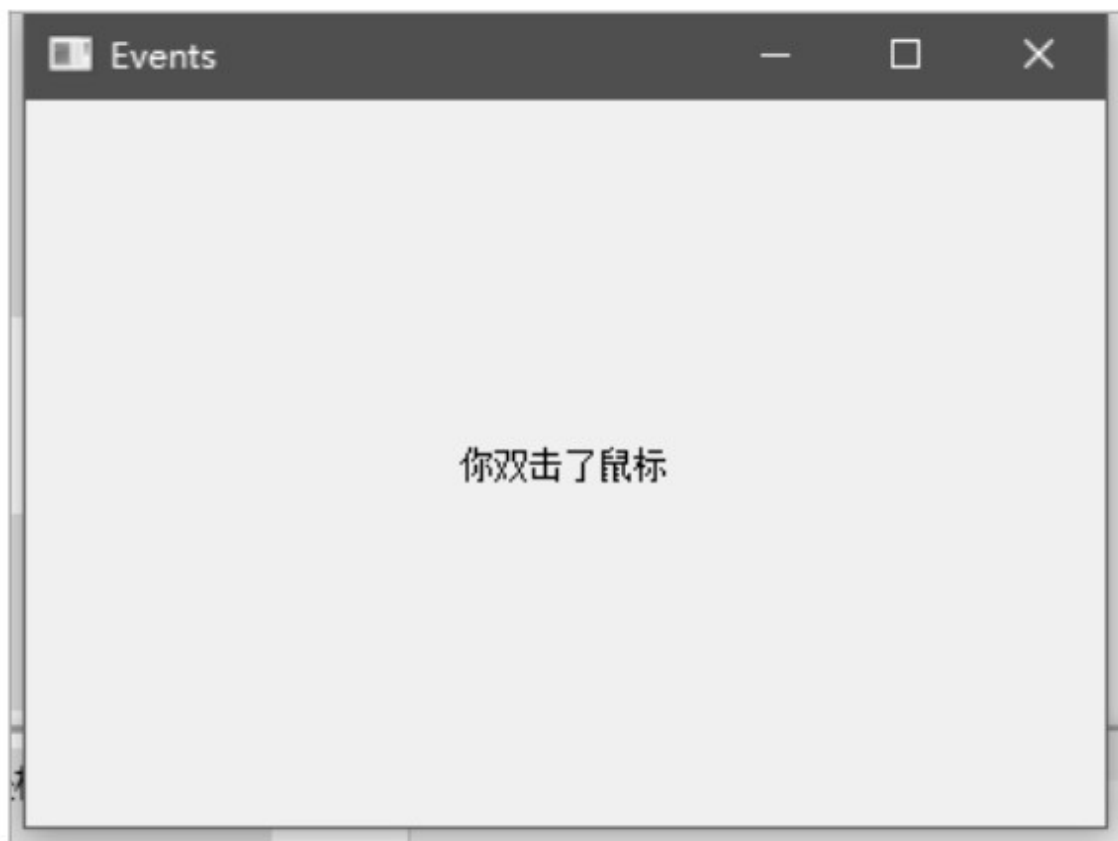


图7-27

下面将介绍如何设置7-28图

```
'''重新实现键盘按下事件'''  
def keyPressEvent(self, event):  
    self.key = ""  
    if event.key() == Qt.Key_Home:  
        self.key = "Home"  
    elif event.key() == Qt.Key_End:  
        self.key = "End"  
    elif event.key() == Qt.Key_PageUp:  
        if event.modifiers() & Qt.ControlModifier:  
            self.key = "Ctrl+PageUp"  
        else:  
            self.key = "PageUp"  
    elif event.key() == Qt.Key_PageDown:  
        if event.modifiers() & Qt.ControlModifier:  
            self.key = "Ctrl+PageDown"  
        else:  
            self.key = "PageDown"  
    elif Qt.Key_A <= event.key() <= Qt.Key_Z:  
        if event.modifiers() & Qt.ShiftModifier:  
            self.key = "Shift+"  
        self.key += event.text()  
    if self.key:  
        self.key = self.key  
        self.update()  
    else:  
        QWidget.keyPressEvent(self, event)
```



图7-28

```

class Event(QWidget):
    def __init__(self):
        super(Event, self).__init__()
        self.setWindowTitle("Events")
        self.installEventFilter(self)
        self.key = ""
        self.setStyleSheet("background-color: #f0f0f0;")
        self.resize(400, 300)
        self.show()

    def event(self, event):
        if event.type() == QEvent.KeyPress and event.key() == Qt.Key_PageDown:
            self.key = "你按下了: PageDown"
            self.update()
            return True
        return super(Event, self).event(event)

    def paintEvent(self, event):
        painter = QPainter(self)
        painter.drawText(100, 150, 300, 250, Qt.AlignCenter)

    def mouseMoveEvent(self, event):
        pass

    def keyPressEvent(self, event):
        pass

    def Tab(self):
        pass

```

```

'''
    本类是PyQt5的QtWidgets模块中的QWidget类的一个子类，它实现了一个
    简单的鼠标跟踪功能。
'''

```

```

def event(self, event):
    if (event.type() == QEvent.KeyPress and
        event.key() == Qt.Key_Tab):
        self.key = "你按下了: Tab"

```

```
self.update()
return True
return QWidget.event(self,event)
```



图7-29

该例子的完整代码位于PyQt5/Chapter07/event\_filter.py  
文件。

```

# -*- coding: utf-8 -*-
from PyQt5.QtGui import *
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
import sys

class EventFilter(QDialog):
    def __init__(self, parent=None):
        super(EventFilter, self).__init__(parent)
        self.setWindowTitle("事件过滤器")

        self.label1 = QLabel("请点击")
        self.label2 = QLabel("请点击")
        self.label3 = QLabel("请点击")
        self.LabelState = QLabel("test")

        self.image1 = QImage("images/cartoon1.ico")
        self.image2 = QImage("images/cartoon1.ico")
        self.image3 = QImage("images/cartoon1.ico")

        self.width = 600
        self.height = 300

        self.resize(self.width, self.height)

        self.label1.installEventFilter(self)
        self.label2.installEventFilter(self)
        self.label3.installEventFilter(self)

        mainLayout = QGridLayout(self)
        mainLayout.addWidget(self.label1, 500, 0)
        mainLayout.addWidget(self.label2, 500, 1)
        mainLayout.addWidget(self.label3, 500, 2)
        mainLayout.addWidget(self.LabelState, 600, 1)
        self.setLayout(mainLayout)

    def eventFilter(self, watched, event):
        if watched == self.label1: # 只对 label1 的点击事件进行过滤, 重写其行
为, 其他事件会被忽略
            if event.type() == QEvent.MouseButtonPress: # 这里对鼠标按下事
件进行过滤, 重写其行为

```



```

        mouseEvent = QMouseEvent(event)
        if mouseEvent.buttons() == Qt.LeftButton:
            self.LabelState.setText("按下鼠标左键")
        elif mouseEvent.buttons() == Qt.MidButton:
            self.LabelState.setText("按下鼠标中间键")
        elif mouseEvent.buttons() == Qt.RightButton:
            self.LabelState.setText("按下鼠标右键")

        '''转换图片大小'''
        transform = QTransform()
        transform.scale(0.5, 0.5)
        tmp = self.image1.transformed(transform)
        self.labell1.setPixmap(QPixmap.fromImage(tmp))
        if event.type() == QEvent.MouseButtonRelease: # 这里对鼠标释放
事件进行过滤, 重写其行为
            self.LabelState.setText("释放鼠标按键")
            self.labell1.setPixmap(QPixmap.fromImage(self.image1))
        return QDialog.eventFilter(self, watched, event) # 对于其他情况,
会返回系统默认的事件处理方法

if __name__ == '__main__':
    app = QApplication(sys.argv)
    dialog = EventFilter()
    dialog.show()
    app.exec_()

```

202007-30 2020-07-31

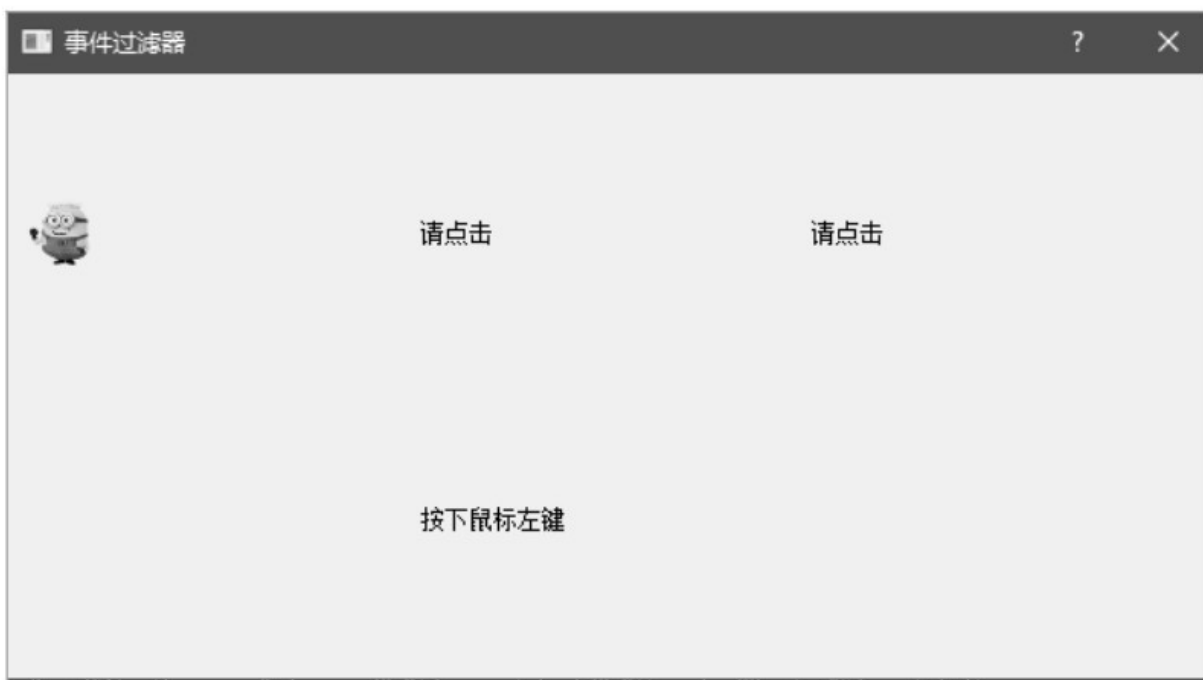


图7-30



图7-31

□□□□□□□□□□□□□□□□

```

class installEventFilter(QObject) eventFilter:
    def installEventFilter(self):
        self.label1.installEventFilter(self)
        self.label2.installEventFilter(self)
        self.label3.installEventFilter(self)

    def eventFilter(self, watched, event):
        if event.type() == QEvent.MouseButtonPress:
            self.label1.setText("按下鼠标左键")
        elif event.type() == QEvent.MouseButtonRelease:
            self.label1.setText("释放鼠标按钮")

```

```

def eventFilter(self, watched, event):
    if watched == self.label1: # 只对label1的点击事件进行过滤，重写其行为，
        # 其他事件会被忽略
        if event.type() == QEvent.MouseButtonPress: # 这里对鼠标按下事件进行过滤，重写其行为
            mouseEvent = QMouseEvent(event)
            if mouseEvent.buttons() == Qt.LeftButton:
                self.LabelState.setText("按下鼠标左键")
            elif mouseEvent.buttons() == Qt.MidButton:
                self.LabelState.setText("按下鼠标中间键")
            elif mouseEvent.buttons() == Qt.RightButton:
                self.LabelState.setText("按下鼠标右键")

            '''转换图片大小'''
            transform = QTransform()
            transform.scale(0.5, 0.5)
            tmp = self.image1.transformed(transform)

```

```

        self.label1.setPixmap(QPixmap.fromImage(tmp))
        if event.type() == QEvent.MouseButtonRelease: # 这里对鼠标释放事件进行过滤，重写其行为
            self.LabelState.setText("释放鼠标按钮")
            self.label1.setPixmap(QPixmap.fromImage(self.image1))
        return QDialog.eventFilter(self, watched, event) # 对于其他情况，会返回系统默认的事件处理方法

```

```

4
'''
transform=QTransform()
transform.scale(0.5,0.5)
tmp=self.image1.transformed(transform)
self.label1.setPixmap(QPixmap.fromImage(tmp))
4
label1
QApplication
'''
label.installEventFilter(self)
# self.label1.installEventFilter(self)
# self.label2.installEventFilter(self)
# self.label3.installEventFilter(self)
QApplication.installEventFilter(dialog)
eventFilter
if __name__ == '__main__':
    app=QApplication(sys.argv)
    dialog=EventFilter()
    app.installEventFilter(dialog)
    dialog.show()
    app.exec_()
PyQt5/Chapter07/event_filter2.py
7-30 7-31
eventFilter
def eventFilter(self,watched,event):
    print(type(watched))
cmd

```

```
class 'PyQt5.QtGui.QWindow'[]
[]class 'PyQt5.QtWidgets.QWidget'[]
[]class 'PyQt5.QtWidgets.QWidget'[]
[]class 'PyQt5.QtWidgets.QWidget'[]
[]class 'PyQt5.QtGui.QWindow'[]
[]class 'PyQt5.QtWidgets.QWidget'[]
[]class 'PyQt5.QtWidgets.QWidget'[]
[]class 'PyQt5.QtWidgets.QWidget'[]
[]class 'PyQt5.QtWidgets.QWidget'[]
[]class 'PyQt5.QtGui.QWindow'[]
[]class '__main__.EventFilter'[]
[]class '__main__.EventFilter'[]
[]class 'PyQt5.QtGui.QWindow'[]
[]class '__main__.EventFilter'[]
[]class '__main__.EventFilter'[]
[]class 'PyQt5.QtWidgets.QLabel'[]
[]class '__main__.EventFilter'[]
[]class 'PyQt5.QtWidgets.QLabel'[]
[]class '__main__.EventFilter'[]
[]class 'PyQt5.QtWidgets.QLabel'[]
[]class '__main__.EventFilter'[]
[]class 'PyQt5.QtWidgets.QLabel'[]
[]class '__main__.EventFilter'[]
[]class '__main__.EventFilter'[]
[]class '__main__.EventFilter'[]
[]class 'PyQt5.QtWidgets.QLabel'[]
```

```
class 'PyQt5.QtWidgets.QLabel'
```

```
    def __init__(self, text, parent=None):
```

```
        super().__init__(parent)
```

## 7.5 信号槽

在 PyQt5 中，信号槽机制是 Qt 框架的核心特性之一。它允许一个对象（信号发送者）发出信号，另一个对象（信号接收者）响应该信号。这种机制实现了对象之间的松耦合，使得程序结构更加清晰和灵活。信号槽机制广泛应用于 GUI 应用程序中，例如按钮点击触发窗口操作、文本输入触发数据更新等。

### 7.5.1 信号槽的基本用法

在 PyQt5 中，使用信号槽的基本步骤如下：

```
1. 创建信号发送者对象（如 QPushButton）和信号接收者对象（如 QMainWindow）。
2. 使用 connect() 方法将发送者的信号与接收者的槽函数连接起来。
3. 运行程序，当信号发出时，槽函数将被自动调用。
```

```

import sys
from PyQt5.QtWidgets import
QWidget,QLCDNumber,QSlider,QVBoxLayout,QApplication
from PyQt5.QtCore import Qt

class WinForm(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        # 先创建滑块和 LCD 控件

        lcd = QLCDNumber(self)
        slider = QSlider(Qt.Horizontal, self)

        vBox = QVBoxLayout()
        vBox.addWidget(lcd)
        vBox.addWidget(slider)

        self.setLayout(vBox)
        # valueChanged() 是 QSlider 的一个信号函数，只要 slider 的值发生改变，它
        就会发射一个信号，然后通过 connect 连接信号的接收控件，也就是 lcd
        slider.valueChanged.connect(lcd.display)

        self.setGeometry(300,300,350,150)
        self.setWindowTitle("信号与槽：连接滑块 LCD")

if __name__ == '__main__':
    app = QApplication(sys.argv)
    form = WinForm()

    form.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□7-32□□□



图7-32

```

class Window(QWidget):
    def __init__(self):
        lcd=QLCDNumber(self)
        slider=QSlider(Qt.Horizontal,self)
        vbox=QVBoxLayout()
        vbox.addWidget(lcd)
        vbox.addWidget(slider)
        slider.valueChanged().connect(lcd.display())
        self.setLayout(vbox)
        valueChanged()  QSlider  slider  connect
        lcd.display()LCD
        slider.valueChanged.connect(lcd.display)
        valueChanged()  QSlider  sliderPressed()
        sliderMoved()sliderReleased()PyQt

```

## 7.5.2 信号与槽的连接



PyQt5 提供了很多对话框，包括文件对话框、输入对话框、颜色对话框、字体对话框等。本章将介绍如何使用这些对话框。

PyQt5 提供了 `QFileDialog`、`QInputDialog`、`QColorDialog`、`QFontDialog` 等对话框类。本章将介绍如何使用这些对话框类。

本章将介绍如何使用 PyQt5/Chapter07/transParam/ DateDialog.py 对话框。

```

from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class DateDialog(QDialog):
    def __init__(self, parent = None):
        super(DateDialog, self).__init__(parent)
        self.setWindowTitle('DateDialog')

        # 在布局中添加控件
        layout = QVBoxLayout(self)
        self.datetime = QDateTimeEdit(self)
        self.datetime.setCalendarPopup(True)
        self.datetime.setDateTime(QDateTime.currentDateTime())
        layout.addWidget(self.datetime)

        # 使用两个按钮(Ok 和 Cancel) 分别连接 accept() 和 reject() 槽函数
        buttons = QDialogButtonBox(
            QDialogButtonBox.Ok | QDialogButtonBox.Cancel,
            Qt.Horizontal, self)
        buttons.accepted.connect(self.accept)
        buttons.rejected.connect(self.reject)
        layout.addWidget(buttons)

        # 从对话框中获取当前日期和时间
        def dateTime(self):
            return self.datetime.dateTime()

        # 使用静态函数创建对话框并返回 (date, time, accepted)
        @staticmethod
        def getDateTime(parent = None):
            dialog = DateDialog(parent)
            result = dialog.exec_()

            date = dialog.dateTime()
            return (date.date(), date.time(), result == QDialog.Accepted)

```

1. Ok/Cancel 버튼을 accept()/reject()로 처리

2. getDateTimes()로 날짜를 가져오기 3. 날짜를 DateDialog로 표시

DateDialog dialog = new DateDialog(this);  
dialog.exec();  
if (dialog.getResult() == QDialogButtonBox.StandardButton.Ok) {  
 // 날짜를 가져오기  
}

CallDialogMainWin.py PyQt5/Chapter07/transParam/CallDialogMainWin.py

```
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from DateDialog import DateDialog

class WinForm(QWidget):

    def __init__(self, parent=None):
        super(WinForm, self).__init__(parent)
        self.resize(400, 90)
        self.setWindowTitle('对话框关闭时返回值给主窗口例子')

        self.lineEdit = QLineEdit(self)
        self.button1 = QPushButton('弹出对话框 1')
        self.button1.clicked.connect(self.onButton1Click)

        self.button2 = QPushButton('弹出对话框 2')
        self.button2.clicked.connect(self.onButton2Click )

        gridLayout = QGridLayout()
        gridLayout.addWidget(self.lineEdit )
        gridLayout.addWidget( self.button1 )
        gridLayout.addWidget( self.button2 )
        self.setLayout(gridLayout)

    def onButton1Click(self ):
        dialog = DateDialog(self)
```

```

        result = dialog.exec_()
        date = dialog.dateTime()
        self.lineEdit.setText( date.date().toString() )
        print('\n 日期对话框的返回值' )
        print('date=%s' % str(date.date()) )
        print('time=%s' % str(date.time()) )
        print('result=%s' % result )
        dialog.destroy()

    def onButton2Click(self ):
        date, time, result = DateDialog.getDateTime()
        self.lineEdit.setText( date.toString() )
        print('\n 日期对话框的返回值' )
        print('date=%s' % str(date) )
        print('time=%s' % str(time) )
        print('result=%s' % result )

if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = WinForm()
    form.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□7-33□□7-34□□□

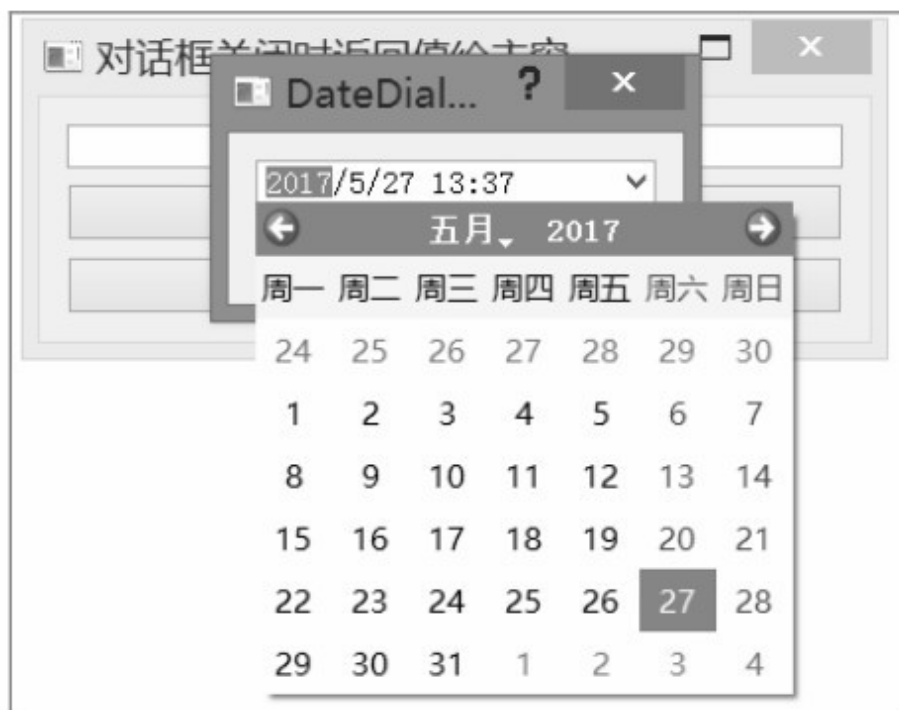


图7-33



图7-34

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class DialogTest {
    public static void main(String[] args) {
        JFrame frame = new JFrame("对话框关闭时返回值给主窗...");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);

        JTextField textField = new JTextField("周六 5月 27 2017");
        JButton button1 = new JButton("弹出对话框1");
        JButton button2 = new JButton("弹出对话框2");

        frame.add(textField);
        frame.add(button1);
        frame.add(button2);

        frame.setVisible(true);
    }
}

```

```
def onButton1Click(self ):
    dialog=DateDialog(self)
    result=dialog.exec_()
    date=dialog.dateTime()
    self.lineEdit.setText( date.date().toString() )
    dialog.destroy()

#####

#####

def onButton2Click(self ):
    date,time,result=DateDialog.getDateTime()
    self.lineEdit.setText( date.toString() )
    if result==QDialog.Accepted :
        print('#####')
    else :
        print('#####')
```

### 7.5.3 九九九九九九九九九九

```

import sys
from PyQt5.QtWidgets import QApplication, QDialog, QVBoxLayout, QWidget, QLabel, QLineEdit, QPushButton, QDateEdit, QDate, QCalendarWidget
from PyQt5.QtCore import Qt, QDate, QDateTime
from PyQt5.QtGui import QIcon

class DateDialog2(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowTitle("Date Dialog")
        self.setModal(True)

        self.layout = QVBoxLayout()
        self.setLayout(self.layout)

        self.date_label = QLabel("Date")
        self.date_edit = QDateEdit()
        self.date_edit.setDate(QDate.currentDate())

        self.ok_button = QPushButton("OK")
        self.cancel_button = QPushButton("Cancel")

        self.layout.addWidget(self.date_label)
        self.layout.addWidget(self.date_edit)
        self.layout.addWidget(self.ok_button)
        self.layout.addWidget(self.cancel_button)

        self.ok_button.clicked.connect(self.ok_clicked)
        self.cancel_button.clicked.connect(self.cancel_clicked)

    def ok_clicked(self):
        date = self.date_edit.date()
        self.close()

    def cancel_clicked(self):
        self.close()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    dialog = DateDialog2()
    dialog.show()
    sys.exit(app.exec_())

```

```
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

class DateDialog(QDialog):
    Signal OneParameter = pyqtSignal(str)
```



```

def __init__(self, parent=None):
    super(DateDialog, self).__init__(parent)
    self.setWindowTitle('子窗口：用来发射信号')

    # 在布局中添加控件
    layout = QVBoxLayout(self)

    self.label = QLabel(self)
    self.label.setText('前者发射内置信号\n后者发射自定义信号')

    self.datetime_inner = QDateTimeEdit(self)
    self.datetime_inner.setCalendarPopup(True)
    self.datetime_inner.setDateTime(QDateTime.currentDateTime())

    self.datetime_emit = QDateTimeEdit(self)
    self.datetime_emit.setCalendarPopup(True)
    self.datetime_emit.setDateTime(QDateTime.currentDateTime())

    layout.addWidget(self.label)
    layout.addWidget(self.datetime_inner)
    layout.addWidget(self.datetime_emit)

    # 使用两个 button(Ok 和 Cancel) 分别连接 accept() 和 reject() 槽函数
    buttons = QDialogButtonBox(
        QDialogButtonBox.Ok | QDialogButtonBox.Cancel,
        Qt.Horizontal, self)
    buttons.accepted.connect(self.accept)
    buttons.rejected.connect(self.reject)
    layout.addWidget(buttons)

    self.datetime_emit.dateTimeChanged.connect(self.emit_signal)

def emit_signal(self):
    date_str = self.datetime_emit.dateTime().toString()
    self.Signal_OneParameter.emit(date_str)

```

□□□□□□□□□□□□□□□□ CallDialogMainWin2.py □□□□□□□  
 PyQt5/Chapter07/transParam/CallDialogMainWin2.py□□□□□□□

□□

```
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
```

```

from PyQt5.QtWidgets import *
from DateDialog2 import DateDialog

class WinForm(QWidget):
    def __init__(self, parent=None):
        super(WinForm, self).__init__(parent)
        self.resize(400, 90)
        self.setWindowTitle('信号与槽传递参数的示例')

        self.open_btn = QPushButton('获取时间')
        self.lineEdit_inner = QLineEdit(self)
        self.lineEdit_emit = QLineEdit(self)
        self.open_btn.clicked.connect(self.openDialog)

        self.lineEdit_inner.setText('接收子窗口内置信号的时间')
        self.lineEdit_emit.setText('接收子窗口自定义信号的时间')

        grid = QGridLayout()
        grid.addWidget(self.lineEdit_inner)
        grid.addWidget(self.lineEdit_emit)

        grid.addWidget(self.open_btn)
        self.setLayout(grid)

    def openDialog(self):
        dialog = DateDialog(self)
        '''连接子窗口的内置信号与主窗口的槽函数'''
        dialog.datetime_inner.dateTimeChanged.connect(
            self.deal_inner_slot)
        '''连接子窗口的自定义信号与主窗口的槽函数'''
        dialog.Signal_OneParameter.connect(self.deal_emit_slot)
        dialog.show()

    def deal_inner_slot(self, date):
        self.lineEdit_inner.setText(date.toString())

    def deal_emit_slot(self, dateStr):
        self.lineEdit_emit.setText(dateStr)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = WinForm()

```

```
form.show()
sys.exit(app.exec_())
```

图7-35

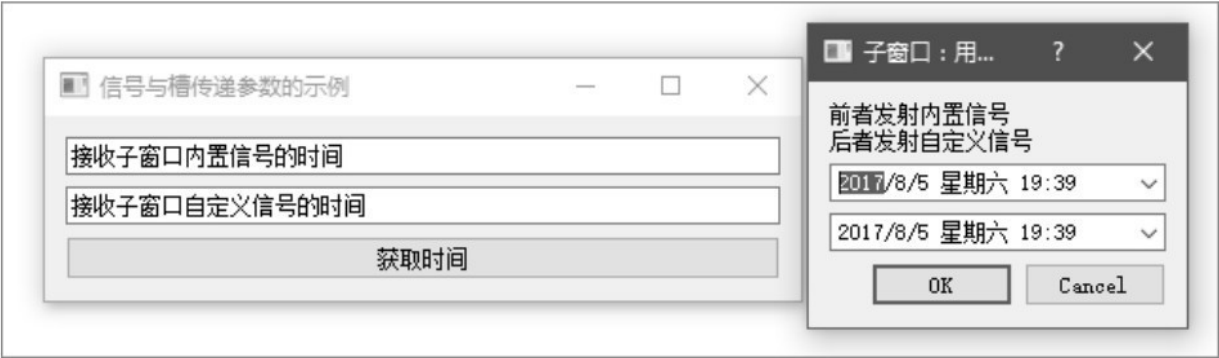


图7-35

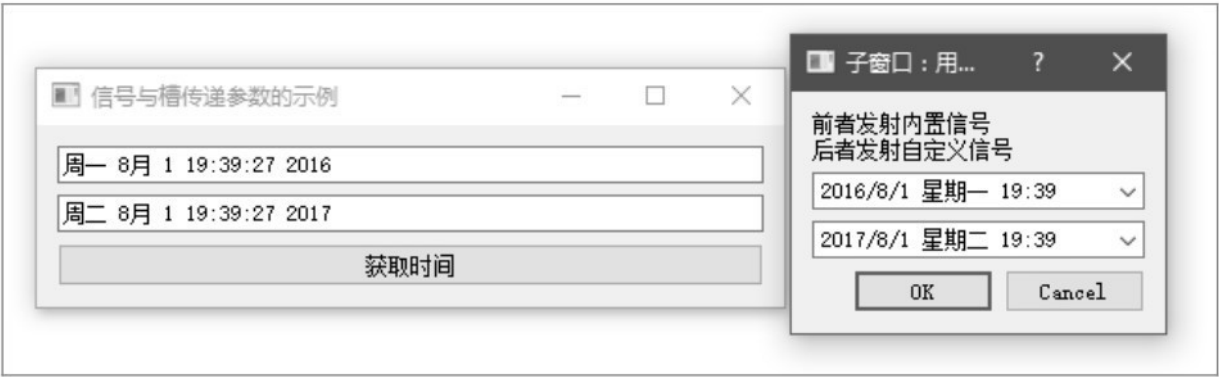


图7-36

```
PyQt5/Chapter07/transParam/DateDialog2.py
datetime_emit
emit_signal
Signal_OneParameter
date_str
self.datetime_emit.dateTimeChanged.connect(self.emit
_signal)
def emit_signal(self):
```



## 第8章 PyQt 5 入门

PyQt 是一个跨平台的 Python 图形用户界面库，它提供了与 Qt 库的接口。Qt 是一个跨平台的 C++ 图形用户界面库，它提供了与操作系统的接口。PyQt 5 支持 Ubuntu、Windows、Mac OS X 等操作系统。QQ 和 360 等软件都是使用 Qt 开发的。PyQt 5 提供了与 Qt 库的接口，使得 Python 程序可以使用 Qt 的图形用户界面库。PyQt 5 提供了与 Qt 库的接口，使得 Python 程序可以使用 Qt 的图形用户界面库。

### 8.1 入门

#### 8.1.1 入门

1. 创建一个 Widget 对象  
setStyle(QStyle style)  
2. 设置样式表  
QStyleFactory.keys()  
3. 使用 QApplication 设置 QStyle  
QApplication.setStyle(QStyleFactory.create("WindowsXP"))  
4. 使用 Widget 设置 QStyle  
QApplication.setStyle(QStyleFactory.create("WindowsXP"))

#### 图 8-1 入门

PyQt5/Chapter08/qt08\_changeStyle.py

```

import sys
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from PyQt5 import QtCore
from PyQt5.QtGui import *

class AppWidget( QWidget):
    def __init__(self, parent=None):
        super(AppWidget, self).__init__(parent)
        horizontalLayout = QHBoxLayout()
        self.styleLabel = QLabel("Set Style:")
        self.styleComboBox = QComboBox()
        # 从 QStyleFactory 中增加多个显示样式
        self.styleComboBox.addItem( QStyleFactory.keys())
        # 选择当前窗口风格
        index = self.styleComboBox.findText(
            QApplication.style().objectName(),
            QtCore.Qt.MatchFixedString)
        # 设置当前窗口风格
        self.styleComboBox.setCurrentIndex(index)
        # 通过 comboBox 控件选择窗口风格
        self.styleComboBox.activated[str].connect
(self.handleStyleChanged)
        horizontalLayout.addWidget(self.styleLabel)
        horizontalLayout.addWidget(self.styleComboBox)
        self.setLayout(horizontalLayout)

        # 改变窗口风格
        def handleStyleChanged(self, style):
            QApplication.setStyle(style)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    widgetApp = AppWidget()
    widgetApp.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□8-1□□□



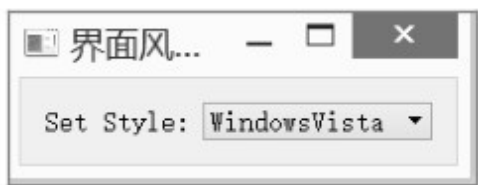


图8-1

## 8.1.2 窗口风格

PyQt 提供了 `setWindowFlags(Qt.WindowFlags)` 函数来设置窗口的风格。该函数的原型如下：

图1 PyQt 窗口风格设置函数

- `Qt.Widget` 窗口风格
- `Qt.Window` 窗口风格
- `Qt.Dialog` 对话框风格
- `Qt.Popup` 弹出式窗口风格
- `Qt.ToolTip` 工具提示风格
- `Qt.SplashScreen` splash screen 风格
- `Qt.SubWindow` 子窗口风格

图2 窗口风格常量

<code>Qt.MSWindowsFixedSizeDialogHint</code>	#窗口无法调整大小
<code>Qt.FramelessWindowHint</code>	#窗口无边框
<code>Qt.CustomizeWindowHint</code>	#有边框但无标题栏和按钮，不能移动和拖动
<code>Qt.WindowTitleHint</code>	#添加标题栏和一个关闭按钮
<code>Qt.WindowSystemMenuHint</code>	#添加系统目录和一个关闭按钮
<code>Qt.WindowMaximizeButtonHint</code>	#激活最大化和关闭按钮，禁止最小化按钮
<code>Qt.WindowMinimizeButtonHint</code>	#激活最小化和关闭按钮，禁止最大化按钮
<code>Qt.WindowMinMaxButtonsHint</code>	#激活最小化、最大化和关闭按钮，相当于
<code>Qt.WindowMaximizeButtonHint   Qt.WindowMinimizeButtonHint</code>	
<code>Qt.WindowCloseButtonHint</code>	#添加一个关闭按钮
<code>Qt.WindowContextHelpButtonHint</code>	#添加问号和关闭按钮，像对话框一样
<code>Qt.WindowStaysOnTopHint</code>	#窗口始终处于顶层位置
<code>Qt.WindowStaysOnBottomHint</code>	#窗口始终处于底层位置

在\_\_init\_\_方法中 self.setWindowFlags() 设置窗口样式  
PyQt5/Chapter08/qt08\_winStyle01.py 窗口无边框

```
from PyQt5.QtCore import Qt
import sys
```

```
from PyQt5.QtWidgets import QMainWindow , QApplication
```

```
class MainWindow(QMainWindow):
    def __init__(self, parent=None):
        super(MainWindow, self).__init__(parent)
        self.resize(400, 200)
        self.setWindowTitle("设置窗口样式例子")
        # 设置无边框窗口样式
        self.setWindowFlags( Qt.FramelessWindowHint )
```

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    win = MainWindow()
    win.show()
    sys.exit(app.exec_())
```

无边框窗口 8-2



图8-2



```
import sys
from PyQt5.QtWidgets import QMainWindow , QApplication
from PyQt5.QtCore import Qt

class MyWindow( QMainWindow):
    '''自定义窗口类'''
def __init__(self,parent=None):
    '''构造函数'''
    # 调用父类构造函数
    super(MyWindow,self).__init__(parent)

    # 设置窗口标志 (无边框)
    self.setWindowFlags( Qt.FramelessWindowHint)

    # 为便于显示, 设置窗口背景颜色 (采用 QSS)
    self.setStyleSheet(''background-color:blue; '')

def showMaximized(self):
    '''最大化窗口'''
    # 得到桌面控件
    desktop = QApplication.desktop()
    # 得到屏幕可显示尺寸
    rect = desktop.availableGeometry()
    # 设置窗口尺寸
    self.setGeometry(rect)
```

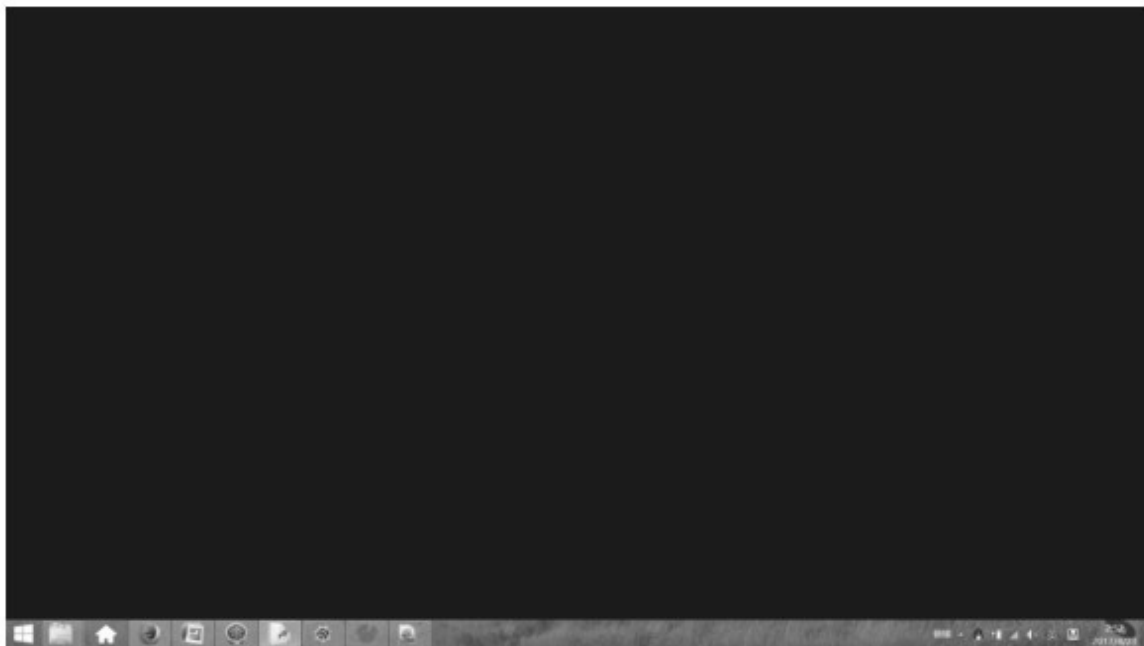
```

        # 显示窗口
        self.show()

# 主函数
if __name__ == "__main__":
    '''主函数'''
    # 声明变量
    app = QApplication(sys.argv)
    # 创建窗口
    window = MyWindow()
    # 调用最大化显示函数
    window.showMaximized()
    # 应用程序事件循环
    sys.exit(app.exec_())

```

□□□□□□□□□□8-3□□□



□8-3

## 8.2 □□

## 8.2.1 PyQt

PyQt 4 QPixmap QImage QPicture QPixmap QBitmap

- QPixmap QPixmap QPixmap
- QImage QImage QImage QImage QImage
- QPicture QPainter QPainter begin() QPicture end() QPicture save() QPainter
- QBitmap QPixmap 1bit QBitmap QPainter QCursor QBrush 8-4

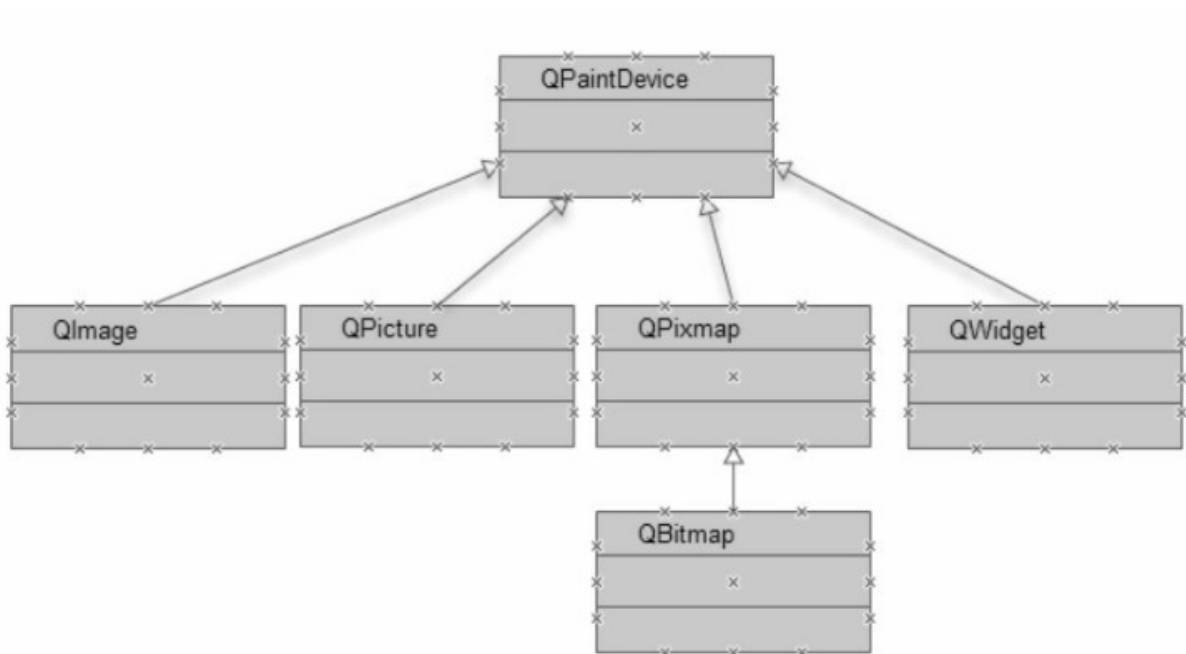


图8-4

## 8.2.2 QPainter

## PyQt5/Chapter08/qt08\_winDraw01.py

```
import sys
from PyQt5.QtWidgets import QApplication ,QWidget
from PyQt5.QtGui import QPainter ,QPixmap
from PyQt5.QtCore import Qt , QPoint

class Winform(QWidget):
    def __init__(self,parent=None):
        super(Winform,self).__init__(parent)
        self.setWindowTitle("绘图例子")
        #1
        self.pix = QPixmap()
        self.lastPoint = QPoint()
        self.endPoint = QPoint()
        self.initUi()

    def initUi(self):
        # 设置窗口大小为 600*500
        self.resize(600, 500)
```

```

        # 设置画布大小为 400*400, 背景为白色
        self.pix = QPixmap(400, 400)
        self.pix.fill(Qt.white)

#2
def paintEvent(self, event):
    pp = QPainter( self.pix)
    # 根据鼠标指针前后两个位置绘制直线
    pp.drawLine( self.lastPoint, self.endPoint)
    # 让前一个坐标值等于后一个坐标值, 就能画出连续的线
    self.lastPoint = self.endPoint
    painter = QPainter(self)
    painter.drawPixmap(0, 0, self.pix)

#3
def mousePressEvent(self, event) :
    # 按下鼠标左键
    if event.button() == Qt.LeftButton :
        self.lastPoint = event.pos()
        self.endPoint = self.lastPoint

#4
def mouseMoveEvent(self, event):
    # 然后移动鼠标指针
    if event.buttons() and Qt.LeftButton :
        self.endPoint = event.pos()
        # 进行重新绘制
        self.update()

#5
def mouseReleaseEvent( self, event):
    # 释放鼠标左键
    if event.button() == Qt.LeftButton :
        self.endPoint = event.pos()
        # 进行重新绘制
        self.update()

if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = Winform()
    form.show()
    sys.exit(app.exec_())

```



图8-5



图8-5

图8-5

图8-5

图8-5

图8-5

图8-5

图8-5

4. `mouseMoveEvent()` 関数をオーバーライドして、マウスが移動したときに呼び出されます。

5. `mouseReleaseEvent()` 関数をオーバーライドして、マウスがリリースされたときに呼び出されます。この関数では、`buttons()` メソッドを使用して、マウスがリリースされたときに押されたボタンを確認します。また、`Qt.LeftButton` 定数を使用して、左ボタンが押されたかどうかを確認します。この関数では、`update()` メソッドを呼び出して、ウィンドウを再描画します。

このようにして、マウスが移動したときにウィンドウの描画を更新し、マウスがリリースされたときに描画を更新することができます。

### 8.2.3 練習問題

この練習問題は、マウスが移動したときにウィンドウの描画を更新し、マウスがリリースされたときに描画を更新するものです。

#### 図8-2 マウスが移動したときにウィンドウの描画を更新する様子

この練習問題は、PyQt5/Chapter08/qt08\_winDraw02.py というファイルで実装されています。

```

import sys
from PyQt5.QtWidgets import QApplication ,QWidget
from PyQt5.QtGui import QPainter ,QPixmap
from PyQt5.QtCore import Qt , QPoint

class Winform(QWidget):
    def __init__(self,parent=None):
        super(Winform,self).__init__(parent)
        self.setWindowTitle("绘制矩形例子")
        self.pix = QPixmap()
        self.lastPoint = QPoint()
        self.endPoint = QPoint()
        self.initUi()

    def initUi(self):
        # 设置窗口大小为 600*500
        self.resize(600, 500)
        # 设置画布大小为 400*400, 背景为白色
        self.pix = QPixmap(400, 400)
        self.pix.fill(Qt.white)

#1
    def paintEvent(self,event):
        painter = QPainter(self)
        x = self.lastPoint.x()
        y = self.lastPoint.y()
        w = self.endPoint.x() - x
        h = self.endPoint.y() - y

        pp = QPainter(self.pix)
        pp.drawRect(x, y, w, h)
        painter.drawPixmap(0, 0, self.pix)

    def mousePressEvent(self, event) :
        # 按下鼠标左键
        if event.button() == Qt.LeftButton :
            self.lastPoint = event.pos()

```

```

        self.endPoint = self.lastPoint

def mouseMoveEvent(self, event):
    # 然后移动鼠标指针
    if event.buttons() and Qt.LeftButton :
        self.endPoint = event.pos()
        # 进行重新绘制
        self.update()

def mouseReleaseEvent( self, event):
    # 释放鼠标左键
    if event.button() == Qt.LeftButton :
        self.endPoint = event.pos()
        # 进行重新绘制
        self.update()

if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = Winform()
    form.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□8-6□□□

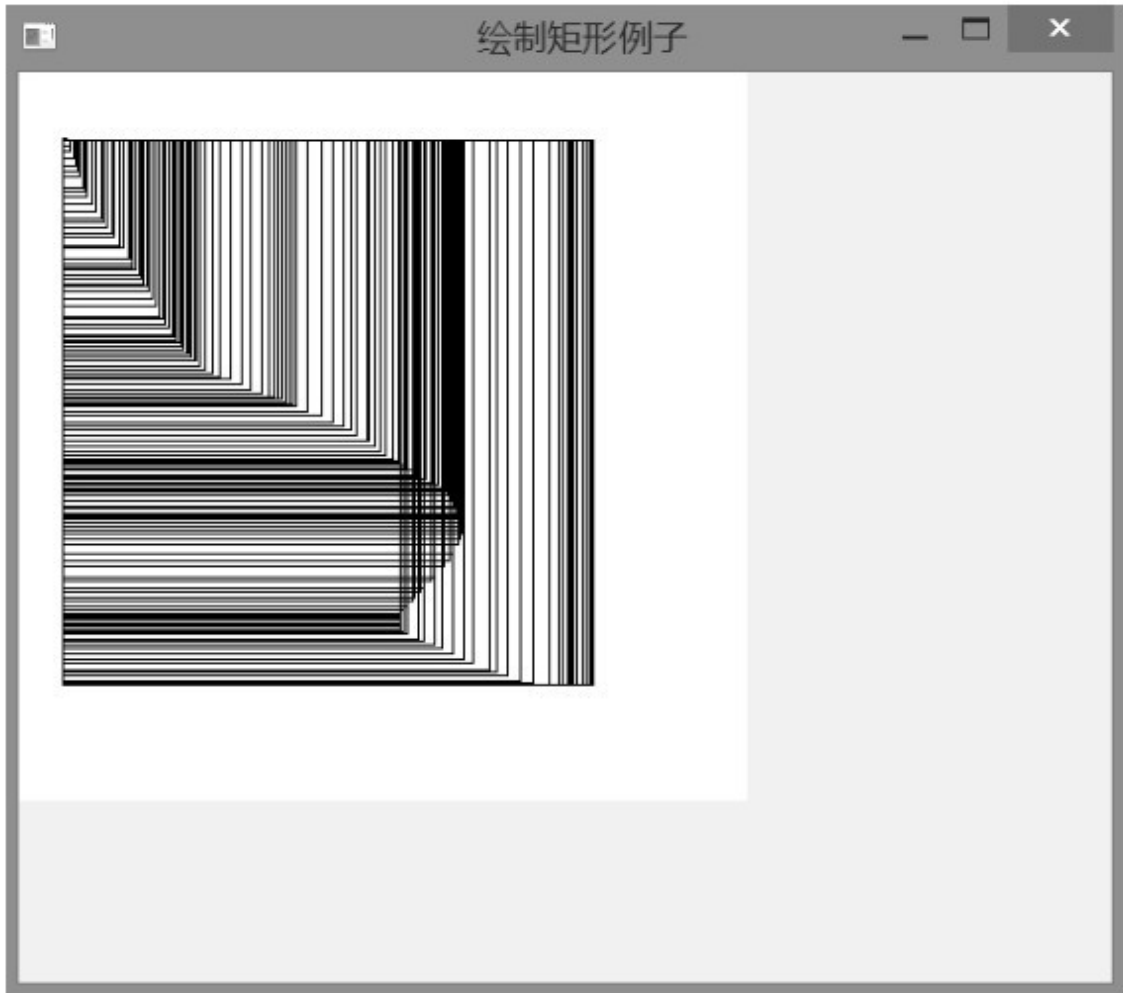


图8-6

```

import java.awt.*;
import javax.swing.*;

public class DrawingRectangles {
    public static void main(String[] args) {
        JFrame frame = new JFrame("绘制矩形例子");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 400);
        frame.getContentPane().setBackground(Color.LIGHT_GRAY);

        JPanel panel = new JPanel();
        panel.setBackground(Color.WHITE);
        frame.add(panel);

        panel.paintEvent(new PaintEvent(panel, 0, 0, panel.getWidth(), panel.getHeight()));
    }
}

```

[图8-3 绘制矩形的代码](#)

## PyQt5/Chapter08/qt08\_winDraw03.py

```
import sys
from PyQt5.QtWidgets import QApplication ,QWidget
from PyQt5.QtGui import QPainter ,QPixmap
from PyQt5.QtCore import Qt , QPoint

class Winform(QWidget):
    def __init__(self,parent=None):
        super(Winform,self).__init__(parent)
        self.setWindowTitle("双缓冲绘图例子")
        self.pix = QPixmap()
        self.lastPoint = QPoint()
        self.endPoint = QPoint()
        #1
        # 辅助画布
        self.tempPix = QPixmap()
        # 标志是否正在绘图
        self.isDrawing = False
        self.initUi()

    def initUi(self):
        # 设置窗口大小为 600*500
        self.resize(600, 500);
        # 设置画布大小为 400*400, 背景为白色
        self.pix = QPixmap(400, 400);
        self.pix.fill(Qt.white);

        #2
    def paintEvent(self,event):
```

```

        painter = QPainter(self)
        x = self.lastPoint.x()
        y = self.lastPoint.y()
        w = self.endPoint.x() - x
        h = self.endPoint.y() - y

        # 如果正在绘图, 就在辅助画布上绘制
        if self.isDrawing :
            # 将以前 pix 中的内容复制到 tempPix 中, 保证以前的内容不消失
            self.tempPix = self.pix
            pp = QPainter( self.tempPix)
            pp.drawRect(x,y,w,h)
            painter.drawPixmap(0, 0, self.tempPix)
        else :
            pp = QPainter(self.pix )
            pp.drawRect(x, y, w, h)
            painter.drawPixmap(0, 0, self.pix)

#3
def mousePressEvent(self, event) :
    # 按下鼠标左键
    if event.button() == Qt.LeftButton :
        self.lastPoint = event.pos()
        self.endPoint = self.lastPoint
        self.isDrawing = True

#4
def mouseReleaseEvent( self, event):
    # 释放鼠标左键
    if event.button() == Qt.LeftButton :
        self.endPoint = event.pos()
        # 进行重新绘制
        self.update()
        self.isDrawing = False

if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = Winform()
    form.show()
    sys.exit(app.exec_())

```

双缓冲绘图例子8-7

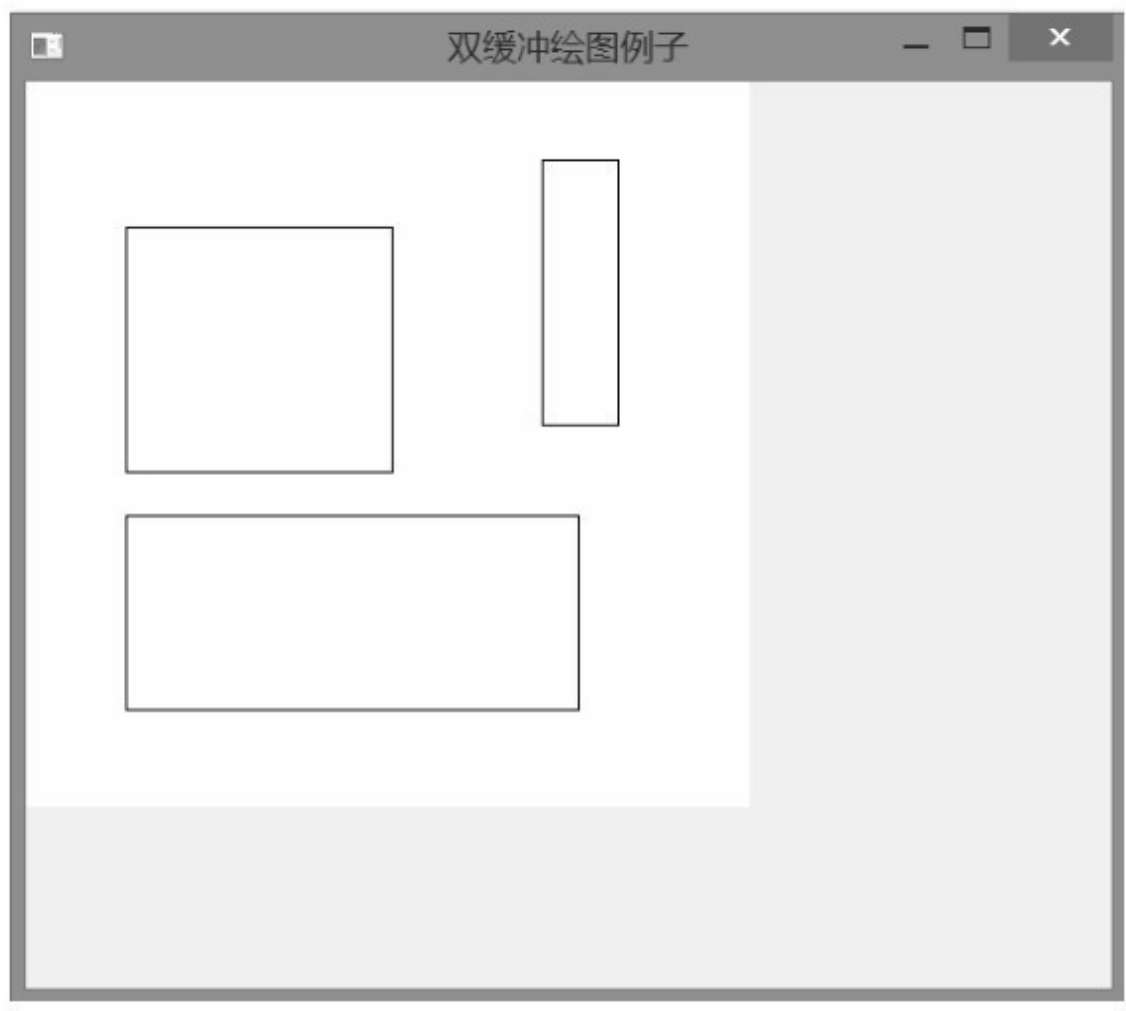


图8-7

```

// 双缓冲绘图例子
// 该程序演示了双缓冲绘图技术，用于避免闪烁。
// 它包含两个缓冲区，交替使用以绘制图形。
// 1. 初始化缓冲区
// # 定义缓冲区大小
```





QPushButton {color:red}  
QPushButton {color:red}

QPushButton {color:red}

QPushButton QPushButton QPushButton  
QPushButton QPushButton QPushButton  
QPushButton QPushButton CSS QPushButton CSS  
{color:red} QPushButton  
PyQt5/Chapter08/qt08\_qssStyle01.py

```
from PyQt5.QtWidgets import *  
import sys  
  
class WindowDemo(QWidget):  
    def __init__(self):  
        super().__init__()  
  
        btn1 = QPushButton(self)  
        btn1.setText('按钮 1')  
  
        btn2 = QPushButton(self)
```

```

        btn2.setText('按钮 2')

        vbox=QVBoxLayout()
        vbox.addWidget(btn1)
        vbox.addWidget(btn2)
        self.setLayout(vbox)
        self.setWindowTitle("QSS 样式")

if __name__ == "__main__":
    app = QApplication(sys.argv)
    win = WindowDemo()
    qssStyle = '''
        QPushButton {
            background-color: red
        }

        '''
    win.setStyleSheet( qssStyle )
    win.show()
    sys.exit(app.exec_())

```

图8-8

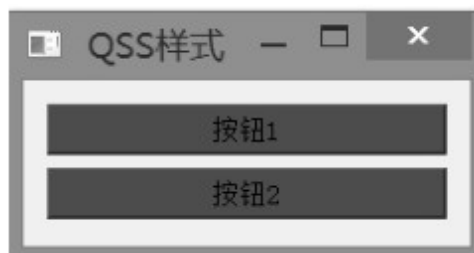


图8-8

图8-8

图8-8展示了QSS样式的应用效果。

图8-8展示了QSS样式的应用效果。win.setStyleSheet()方法用于设置QSS样式。

图8-8展示了QSS样式的应用效果。setStyleSheet()方法用于设置QSS样式。

图8-8展示了QSS样式的应用效果。



win.show()

图8-9 图8-9 “图2” 图8-9



图8-9

4 QPushButton QPushButton  
CSS

5 ID #myButton ID myButton ID  
objectName

6 QDialog QPushButton QDialog  
QPushButton

7 QDialog QPushButton QDialog  
QPushButton QPushButton QDialog

#frameCut, #frameInterrupt, #frameJoin ID  
#mytable QPushButton ID mytable  
QPushButton

### 8.3.3 QSS

QSS QComboBox  
QComboBox::drop-down { image: url(dropdown.png) }



```

        combo.addItem('Red Hat')
        combo.move(50,50)
        self.setGeometry(250,200,320,150)
        self.setWindowTitle('QComboBox 样式')

if __name__ == "__main__":
    app = QApplication(sys.argv)
    win = WindowDemo()
    # 定义 QComboBox 控件的 QSS 样式
    qssStyle = '''
        QComboBox#myQComboBox::drop-down {
            image: url( ./images/dropdown.png)
        }

        '''
    win.setStyleSheet( qssStyle )
    win.show()
    sys.exit(app.exec_())

```

图 8-10



图 8-10

### 8.3.4 QSS

QSS 是 Qt 提供的一种样式表语言，它类似于 CSS。QSS 提供了一种简单、灵活的方式来定义 Qt 控件的外观。QSS 可以应用于 Qt 控件的实例，也可以应用于 Qt 控件的类。QSS 可以定义控件的字体、颜色、背景、边框、图标等属性。QSS 还可以定义控件的悬停（hover）状态、禁用（disabled）状态、焦点（focus）状态等。QSS 还可以定义控件的动画效果。QSS 的语法非常简单，易于学习和使用。QSS 是 Qt 中实现自定义控件外观的重要工具。

```
QComboBox:hover{background-color:red;}
```

QComboBox :hover  
QComboBox  
QComboBox

```
QComboBox::drop-down:hover{background-color:red;}
```

```

QComboBox
: hover
:! hover

```

```
QCheckBox:hover:checked { color: white }
```

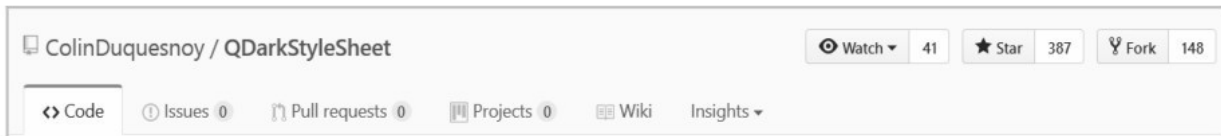
PyQt5 QCheckBox PyQt5 QSS PyQt5 PyQt

### 8.3.5 QDarkStyleSheet

QDarkStyleSheetPyQtGitHub

<https://github.com/ColinDuquesnoy/QDarkStyleSheet/tree/master/qdarkstyle>

8-11 QDarkStyleSheet 387



□8-11

## 1. QDarkStyleSheet



Click the “Clone or download” button to clone QDarkStyleSheet to your local machine. The path is PyQt5\Chapter08\QDarkStyleSheet-master. Figure 8-12 shows the clone process.

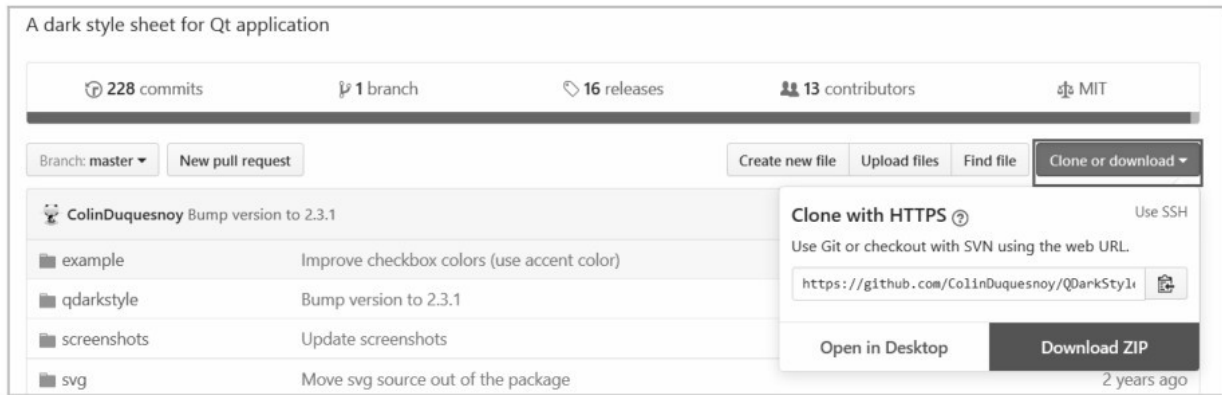


Figure 8-12

```
pip install qdarkstyle
2. Clone QDarkStyleSheet
from qdarkstyle import qdarkstyle
app.setStyleSheet(qdarkstyle.load_stylesheet_pyqt5())
PyQt5\Chapter08\QDarkStyleSheet-master\example\ example_pyqt5.py
```

```
import logging
import sys
from PyQt5 import QtWidgets, QtCore
# make the example runnable without the need to install
from os.path import abspath, dirname
sys.path.insert(0, abspath(dirname(abspath(__file__))) + '/../..'))

import qdarkstyle
import ui.example_pyqt5_ui as example_ui

def main():
    """
    Application entry point
    """
    logging.basicConfig(level=logging.DEBUG)
    # create the application and the main window
    app = QtWidgets.QApplication(sys.argv)
    window = QtWidgets.QMainWindow()

    # setup ui
    ui = example_ui.Ui_MainWindow()
    ui.setupUi(window)
    ui.bt_delay_popup.addAction([
        ui.actionAction,
        ui.actionAction_C
    ])
    ui.bt_instant_popup.addAction([
        ui.actionAction,
        ui.actionAction_C
    ])
    ui.bt_menu_button_popup.addAction([
        ui.actionAction,
```

```

        ui.actionAction_C
    ])
    item = QtWidgets.QTableWidgetItem("Test")
    item.setCheckState(QtCore.Qt.Checked)
    ui.tableWidget.setItem(0, 0, item)
    window.setWindowTitle("QDarkStyle example")

    # tabify dock widgets to show bug #6
    window.tabifyDockWidget(ui.dockWidget1, ui.dockWidget2)

    # setup stylesheet
    app.setStyleSheet(qdarkstyle.load_stylesheet_pyqt5())

    # auto quit after 2s when testing on travis-ci
    if "--travis" in sys.argv:
        QtCore.QTimer.singleShot(2000, app.exit)

    # run
    window.show()
    app.exec_()

if __name__ == "__main__":
    main()

```

8-13

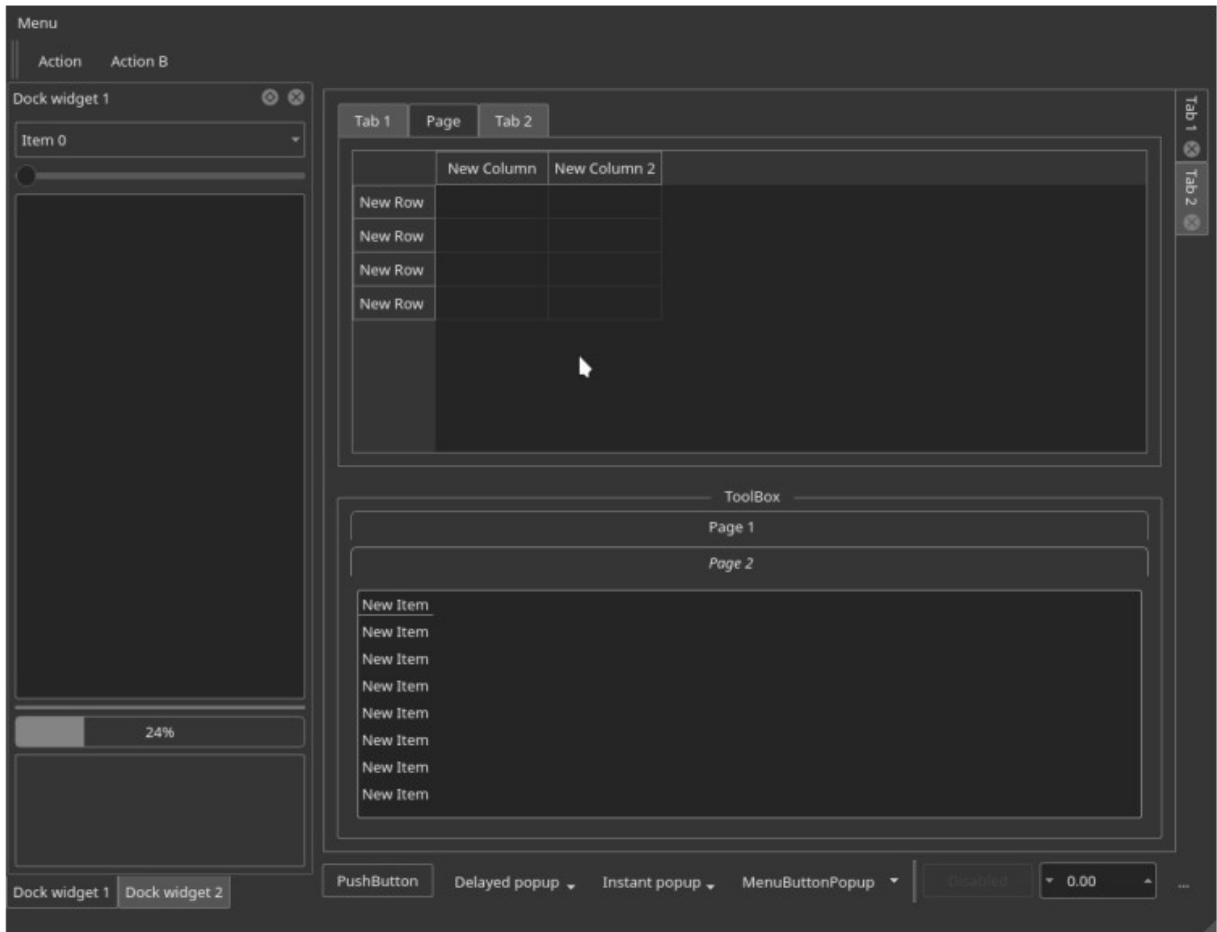


图8-13

## 8.4 自定义UI

本章将介绍如何自定义UI，包括如何自定义UI的布局、如何自定义UI的样式、如何自定义UI的交互等。

- 使用QSS自定义UI
- 使用QPalette自定义UI
- 使用paintEvent和QPainter自定义UI

### 8.4.1 使用QSS自定义UI

QSS 的 background 和 background-color 属性  
setPixmap 和 setIcon 属性  
4-7 和 4-14

### 1. `setStyleSheet()`

`MainWindow` 的 `setStyleSheet()`  
PyQt5/Chapter08/qt08\_winBkgground01.py

```
win=QMainWindow()
# 
win.setObjectName("MainWindow")
# 
win.setStyleSheet("#MainWindow{border-
image:url(images/python.jpg);}")
win.setStyleSheet("#MainWindow{border-
image:url(e:/images/python.jpg);}")
8-14
```



图8-14

## 2. 使用setStyleSheet()设置背景色

```
win=QMainWindow()  
# 设置背景色  
win.setObjectName("MainWindow")  
win.setStyleSheet("#MainWindow{background-color:  
yellow}")
```

图8-15



图8-15

## 8.4.2 使用QPalette设置背景色

### 1. 使用QPalette设置背景色

文件路径: PyQt5/Chapter08/qt08\_winBkgground03.py

代码

```
win=QMainWindow()  
palette=QPalette()  
palette.setColor(QPalette.Background ,Qt.red )  
win.setPalette(palette)
```





图8-17

1 在代码中调用 `setPalette()` 方法设置调色板  
 文件: `PyQt5/Chapter08/qt08_winBkground02.py`  

```
win=QMainWindow()
palette=QPalette()
```



```

palette.setBrush(QPalette.Background,QBrush(QPixmap(
p("./images/python.jpg")))
win.setPalette(palette)
win.resize(460,255 )

```

界面背景图片设置8-18



图8-18

```

2
palette=QPalette()
palette.setBrush(QPalette.Background,QBrush(QPixmap(
p("./images/python.jpg")))
win.setPalette(palette)
win.resize(800,600 )

```

界面背景图片设置8-19



图8-19

### 8.4.3 实现paintEvent函数

#### 1. 实现paintEvent函数

在文件PyQt5/Chapter08/qt08\_winBkgground04.py中添加如下

代码

```
class Winform(QWidget):
    def __init__(self,parent=None):
        super(Winform,self).__init__(parent)
        self.setWindowTitle("paintEvent函数")
    def paintEvent(self,event):
```

```

painter=QPainter(self)
painter.setBrush(Qt.black );
# 填充
painter.drawRect( self.rect());
"""8-20"""

```

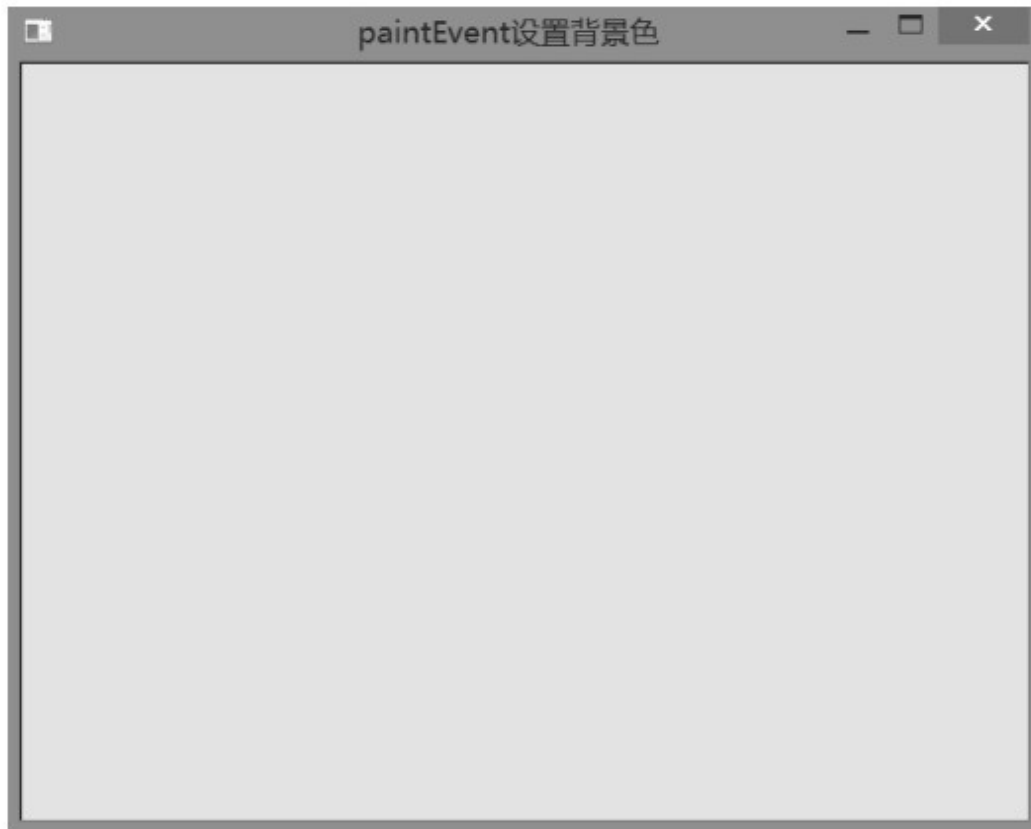


图8-20

## 2. 实现paintEvent函数

文件路径PyQt5/Chapter08/qt08\_winBkgground05.py

"""

```

class Winform(QWidget):
    def __init__(self,parent=None):
        super(Winform,self).__init__(parent)

```

```

self.setWindowTitle("paintEvent设置背景")
def paintEvent(self,event):
    painter=QPainter(self)
    pixmap=QPixmap("./images/screen1.jpg")
    #将pixmap绘制到窗口
    painter.drawPixmap(self.rect(),pixmap)

```

运行结果如图8-21所示



图8-21

## 8.5 使用QPainter

QWidget类提供了paintEvent()方法，用于绘制图形。图8-1所示

图8-1

函 数	描 述
setMask(self, QBitmap) setMask(self, QRegion)	setMask()的作用是为调用它的控件增加一个遮罩，遮住所选区域以外的部分，使之看起来是透明的。它的参数可以为 QBitmap 或 QRegion 对象，此处调用 QPixmap 的 mask()函数获得图片自身的遮罩，是一个 QBitmap 对象。在示例中使用的是 PNG 格式的图片，它的透明部分实际上就是一个遮罩
paintEvent(self, QPaintEvent)	通过重载 paintEvent()函数绘制窗口背景

图 1 展示了使用 QPainter 类中的 drawImage() 函数绘制图片的示例。在 paintEvent() 函数中，通过调用 QPainter 类的 drawImage() 函数，将图片绘制到窗口背景上。图 8-22 展示了使用 QPainter 类中的 drawImage() 函数绘制图片的示例。



图8-22

该示例位于 PyQt5/Chapter08/qt08\_paint01.py 文件中。

```

import sys
from PyQt5.QtWidgets import QApplication ,QWidget
from PyQt5.QtGui import QPixmap, QPainter , QPixmap

class MyForm(QWidget):
    def __init__(self,parent=None):
        super(MyForm,self).__init__(parent)
        self.setWindowTitle("不规则窗口的实现例子")

```

```

    def paintEvent(self,event):
        painter = QPainter(self)
        painter.drawPixmap(0,0,280,390,QPixmap(r"./images/dog.jpg"))
        painter.drawPixmap(300,0,280,390,QPixmap
(r"./images/dog.jpg"))

if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = MyForm()
    form.show()
    sys.exit(app.exec_())

```

000000000000008-230000



图8-23

图2展示了在Windows窗体中实现不规则窗体的过程。在paintEvent()方法中，通过调用mask.png和screen1.jpg文件，结合图8-24和图8-25所示的代码，实现了窗体的不规则形状。



图8-24



图8-25

该图由PyQt5/Chapter08/qt08\_paint02.py生成



```

import sys
from PyQt5.QtWidgets import QApplication ,QWidget
from PyQt5.QtGui import QPixmap, QPainter , QBitmap

class Winform(QWidget):
    def __init__(self,parent=None):
        super(Winform,self).__init__(parent)
        self.setWindowTitle("不规则窗口的实现例子")

        self.pix = QBitmap("./images/mask.png")
        self.resize(self.pix.size())
        self.setMask(self.pix)

    def paintEvent(self,event):
        painter = QPainter(self)
        # 在指定区域直接绘制窗口背景
        painter.drawPixmap(0,0,self.pix.width(),self.pix.height(),
        QPixmap("./images/screen1.jpg"))

if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = Winform()
    form.show()
    sys.exit(app.exec_())

```

□ 3 □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □ □  
 PyQt5/Chapter08/qt08\_paint03.py□□□□□□□□

```
import sys
from PyQt5.QtWidgets import QApplication ,QWidget
from PyQt5.QtGui import QPixmap, QPainter , QCursor
from PyQt5.QtCore import Qt

class ShapeWidget(QWidget):
    def __init__(self,parent=None):
        super(ShapeWidget,self).__init__(parent)
        self.setWindowTitle("不规则的可以拖动的窗口实现例子")
        self.mypix()

# 显示不规则的图片
def mypix(self):
    self.mypic = './images/boy.jpg'
    self.pix = QPixmap(self.mypic , "0", Qt.AvoidDither |
```

```

Qt.ThresholdDither | Qt.ThresholdAlphaDither)
    self.resize(self.pix.size())
    self.setMask(self.pix.mask())
    self.dragPosition=None

    # 重定义鼠标按键按下响应函数 mousePressEvent (QMouseEvent) 和鼠标指针移动响
    应函数 mouseMoveEvent (QMouseEvent), 使不规则窗口能响应鼠标事件, 随意拖动窗口
    def mousePressEvent(self, event):
        if event.button() == Qt.LeftButton:
            self.m_drag=True
            self.m_DragPosition=event.globalPos()-self.pos()
            event.accept()
            self.setCursor(QCursor(Qt.OpenHandCursor))

    def mouseMoveEvent(self, QMouseEvent):
        if Qt.LeftButton and self.m_drag:
            # 当使用左键移动窗口时修改偏移值
            self.move(QMouseEvent.globalPos()- self.m_DragPosition )
            QMouseEvent.accept()

    def mouseReleaseEvent(self, QMouseEvent):
        self.m_drag=False
        self.setCursor(QCursor(Qt.ArrowCursor))

    # 在窗口中首次绘制时, 会加载 paintEvent() 函数
    def paintEvent(self, event):
        painter = QPainter(self)
        painter.drawPixmap(0, 0,
            self.pix.width(),
            self.pix.height(),
            self.pix )

    # 鼠标双击事件
    def mouseDoubleClickEvent(self, event):
        self.mypix()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    form = ShapeWidget()
    form.show()
    sys.exit(app.exec ())

```

### 8.5.1 遮罩效果

```
#####PyQt#####  
1 pixmap.setMask()#####  
#####QBitmap#####QRegion#####  
QPixmap self.pix.mask()#####QBitmap  
    self.pix=QPixmap(self.mypic, "0", Qt.AvoidDither |  
Qt.ThresholdDither| Qt.ThresholdAlphaDither)  
    self.setMask(self.pix.mask())  
2 paintEvent()#####  
paintEvent()#####  
    self.timer=QTimer()  
    self.timer.setInterval(500)  
    self.timer.timeout.connect(self.timeChange)  
    self.timer.start()  
#####timer#####  
    self.update()  
#####PyQt5/Chapter08/qt08_paint04.py#####
```

```
import sys
from PyQt5.QtWidgets import QApplication ,QWidget
from PyQt5.QtGui import QPixmap, QPainter , QCursor
from PyQt5.QtCore import Qt, QTimer

class ShapeWidget(QWidget):
    def __init__(self,parent=None):
        super(ShapeWidget,self).__init__(parent)
        self.i = 1
        self.mypix()
        self.timer = QTimer()
        self.timer.setInterval(500) # 定时器每 500 毫秒更新一次
        self.timer.timeout.connect(self.timeChange)
        self.timer.start()

# 显示不规则图片
def mypix(self):
```

```

        self.update()
        if self.i == 5:
            self.i = 1
        self.mypic = {1: './images/left.png', 2: './images/up.png', 3:
        './images/right.png', 4: './images/down.png'}
        self.pix = QPixmap(self.mypic[self.i], "0", Qt.AvoidDither |
        Qt.ThresholdDither | Qt.ThresholdAlphaDither)
        self.resize(self.pix.size())
        self.setMask(self.pix.mask())
        self.dragPosition = None

    def mousePressEvent(self, event):
        if event.button() == Qt.LeftButton:
            self.m_drag=True
            self.m_DragPosition=event.globalPos()-self.pos()
            event.accept()
            self.setCursor(QCursor(Qt.OpenHandCursor))

    def mouseMoveEvent(self, QMouseEvent):
        if Qt.LeftButton and self.m_drag:
            self.move(QMouseEvent.globalPos()- self.m_DragPosition )
            QMouseEvent.accept()

    def mouseReleaseEvent(self, QMouseEvent):
        self.m_drag=False
        self.setCursor(QCursor(Qt.ArrowCursor))

    def paintEvent(self, event):
        painter = QPainter(self)
        painter.drawPixmap(0, 0,
self.pix.width(),self.pix.height(),self.pix)

    # 鼠标双击事件
    def mouseDoubleClickEvent(self, event):
        if event.button() == 1:
            self.i += 1
            self.mypix()

    # 每 500 毫秒窗口执行一次更新操作，重绘窗口
    def timeChange(self):
        self.i += 1
        self.mypix()

```

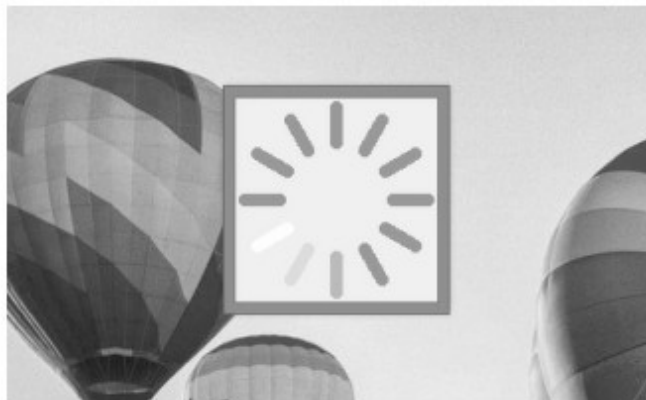


```

self.label=QLabel("",self)
self.setFixedSize(128,128)
self.setWindowFlags(Qt.Dialog|
Qt.CustomizeWindowHint)
self.movie=QMovie("./images/loading.gif")
self.label.setMovie(self.movie)
self.movie.start()
if __name__=='__main__':
    app=QApplication(sys.argv)
    loadingGitWin=LoadingGifWin()
    loadingGitWin.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□8-27□□□



□8-27

## **8.6** □□□□

### **8.6.1** □□□□□□□□□□



PyQt5/Chapter08/qt08\_labelStyle.py

```
label1=QLabel(self)
label1.setToolTip('Python')
label1.setStyleSheet("QLabel{border-image:
url(./images/python.jpg);}")
# Python
label1.setFixedWidth(476)
label1.setFixedHeight(259)
```

### 8.6.2 □□□□□□□□

PyQt5/Chapter08/qt08\_btnStyle.py

```
btn1=QPushButton(self )
btn1.setMaximumSize(48,48)
btn1.setMinimumSize(48,48)
style=""
    QPushButton{
        border-radius: 30px;
        background-image: url('./images/left.png');
    }
    ...

btn1.setStyleSheet(style)

#####QPushButton#####
#####QPushButton#####setObjectName()#####
#####
#####
]
```

```

btn1 = QPushButton(self )
btn1.setObjectName('btn1')
btn1.setMaximumSize(64, 64)
btn1.setMinimumSize(64, 64)
style = '''
    #btn1{
        border-radius: 30px;
        background-image: url('./images/left.png');
    }

    #btn1:Hover{
        border-radius: 30px;
        background-image: url('./images/leftHover.png');
    }

    #btn1:Pressed{
        border-radius: 30px;
        background-image: url('./images/leftPressed.png');
    }

    '''

btn1.setStyleSheet(style)

```

□□□□□□□□□□8-28□□□



图8-28

### 8.6.3 加载图片

下面代码位于PyQt5/Chapter08/qt08\_imgScaled.py文件中

```
# filename 为图片的路径
filename = r".\images\Cloudy_72px.png"
img = QImage( filename )
# 设置标签的宽度为 120 像素，高度为 120 像素，所加载的图片按照标签的高度和宽度等比例
缩放
label1 = QLabel(self)
label1.setFixedWidth(120)
label1.setFixedHeight(120)
# 缩放图片，以固定大小显示
result = img.scaled(label1.width(),
label1.height(),Qt.IgnoreAspectRatio, Qt.SmoothTransformation);
# 在标签控件上显示图片
label1.setPixmap(QPixmap.fromImage(result))
```

下面代码位于PyQt5/Chapter08/qt08\_imgScaled.py文件中



图8-29

#### 8.6.4 窗口透明度

窗口透明度是指窗口在屏幕上的可见程度。窗口的透明度范围从0.0（完全透明）到1.0（完全不透明）。  
 窗口透明度的设置方法如下：

```
win=QMainWindow()
```

```
win.setWindowOpacity(0.5);
```

其中0.0表示完全透明，1.0表示完全不透明。

示例代码位于PyQt5/Chapter08/qt08\_WindowOpacity.py。

图8-30 窗口透明度

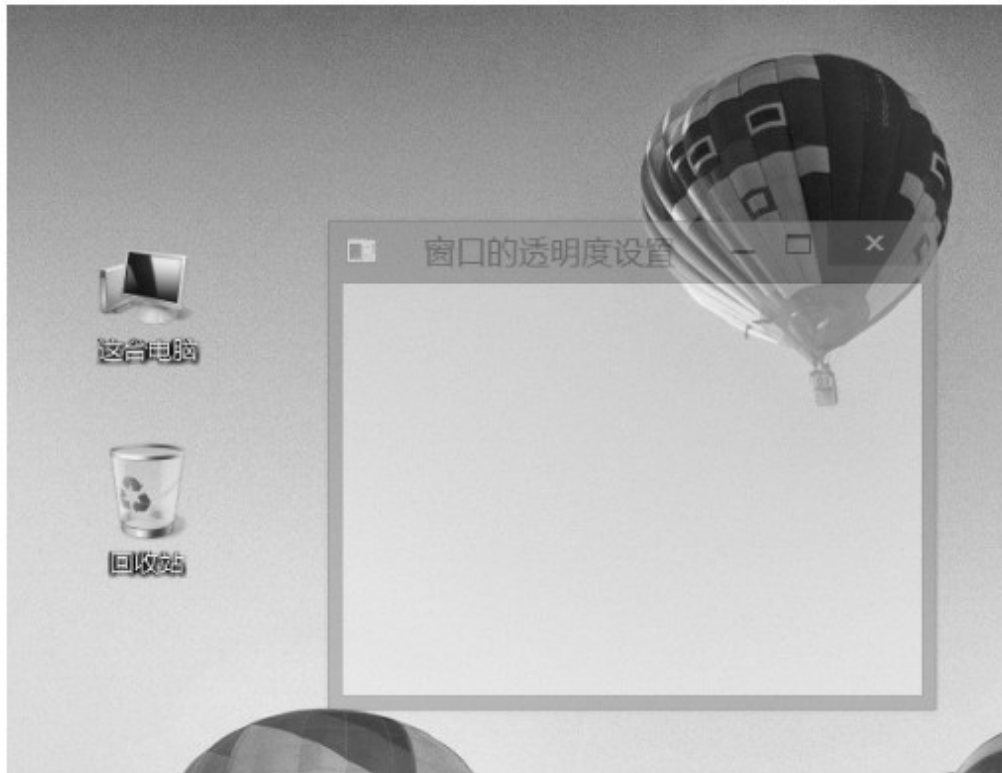


图8-30

### 8.6.5 Qt QSS

Qt 提供了 Qt Style Sheets (QSS) 功能，用于自定义 Qt 应用程序的外观。QSS 类似于 CSS，用于定义 Qt 控件的样式。Qt 提供了 QLabel、QLineEdit、QPushButton 等控件，这些控件都支持 QSS。Qt 应用程序的 QMainWindow 窗口支持 QSS，可以通过 QMainWindow 的 setStyleSheet() 方法设置窗口的样式。

#### 1. Qt QSS

Qt 提供了 Qt Style Sheets (QSS) 功能，用于自定义 Qt 应用程序的外观。QSS 类似于 CSS，用于定义 Qt 控件的样式。Qt 提供了 QLabel、QLineEdit、QPushButton 等控件，这些控件都支持 QSS。Qt 应用程序的 QMainWindow 窗口支持 QSS，可以通过 QMainWindow 的 setStyleSheet() 方法设置窗口的样式。

```
MainWindow{
    border-image:url(./images/python.jpg);
}

QToolTip{
```

```

border: 1px solid rgb(45,45,45);
background: white;
color: red;
}

```

## 2. QSS

CommonHelper PyQt5/Chapter08/CommonHelper.py

```

class CommonHelper :
    def __init__(self ) :
        pass
    @staticmethod
    def readQss( style):
        with open( style , 'r') as f:
            return f.read()

```

PyQt5/Chapter08/qt08\_loadQss.py

```

app=QApplication(sys.argv)
win=MainWindow()
styleFile='./style.qss'
# CommonHelper.readQss()
style=CommonHelper.readQss( styleFile )
win.setStyleSheet( style )
win.show()
sys.exit(app.exec_())

```

CommonHelper.readQss() QSS

8-31



图8-31

下面将图8-32所示



图8-32

## 9 PyQt 5

PyQt5 PyQt  
 Python PyQt Qt  
 Python PyQt Python  
 PyInstaller Pandas Matplotlib PyQtGraph Plotly  
 PyQt  
 PyQt GUI

## 9.1 PyInstaller EXE

PyQt 5  
 .py .py .exe  
 PyInstaller PyQt5 EXE

PyInstallerWindowsLinuxMac OS  
32 64 http://www.pyinstaller.org/  
 PyInstallerPyQt 5 EXE

## 1. Python 3.5+PyQt 5.9

Python 3.5+PyQt 5.9  
PyQt 5.9-1

## 2. PyInstaller



1. pip 安装 PyInstaller  
 运行 EXE 文件需要 pip 安装 PyInstaller  
 运行 pypiwin32 需要 pip 安装 PyInstaller  
 pip install PyInstaller  
 PyInstaller 版本 9.1

图 9-1

名 称	版 本
操作系统	64 位 Windows 8
Python	3.5.3
PyQt	5.9
Eric	6.17

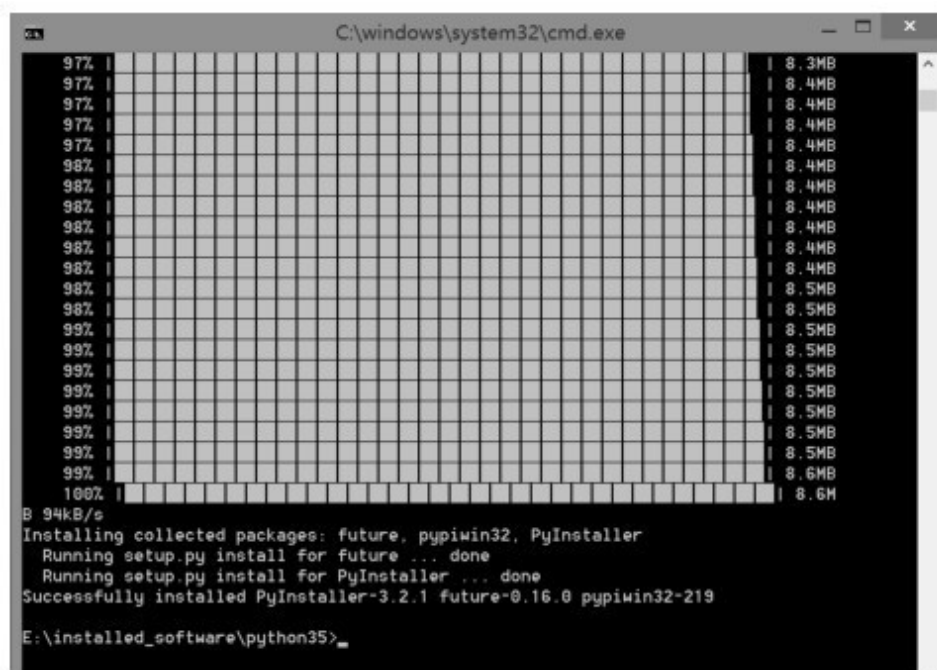


图 9-1

2. PyInstaller 安装 Python Scripts  
 pyinstaller.exe 安装 pip  
 E:\installed\_software\python35\Scripts

图 9-2

这台电脑 > 新加卷 (E:) > installed\_software > python35 > Scripts

名称	修改日期	类型	大小
eric6_sqlbrowser.bat	2017/3/22 18:19	Windows 批处理...	1 KB
eric6_tray.bat	2017/3/22 18:19	Windows 批处理...	1 KB
eric6_trpreviewer.bat	2017/3/22 18:19	Windows 批处理...	1 KB
eric6_uipreviewer.bat	2017/3/22 18:19	Windows 批处理...	1 KB
eric6_unittest.bat	2017/3/22 18:19	Windows 批处理...	1 KB
eric6_webbrowser.bat	2017/3/22 18:19	Windows 批处理...	1 KB
futurize.exe	2017/3/28 0:27	应用程序	73 KB
futurize-script.py	2017/3/28 0:27	Python File	1 KB
pasteurize.exe	2017/3/28 0:27	应用程序	73 KB
pasteurize-script.py	2017/3/28 0:27	Python File	1 KB
pip.exe	2017/3/22 17:43	应用程序	96 KB
pip3.5.exe	2017/3/22 17:43	应用程序	96 KB
pip3.exe	2017/3/22 17:43	应用程序	96 KB
pyi-archive_viewer.exe	2017/3/28 0:28	应用程序	73 KB
pyi-archive_viewer-script.py	2017/3/28 0:28	Python File	1 KB
pyi-bindepend.exe	2017/3/28 0:28	应用程序	73 KB
pyi-bindepend-script.py	2017/3/28 0:28	Python File	1 KB
pyi-grab_version.exe	2017/3/28 0:28	应用程序	73 KB
pyi-grab_version-script.py	2017/3/28 0:28	Python File	1 KB
pyi-makespec.exe	2017/3/28 0:28	应用程序	73 KB
pyi-makespec-script.py	2017/3/28 0:28	Python File	1 KB
pyinstaller.exe	2017/3/28 0:28	应用程序	73 KB
pyinstaller-script.py	2017/3/28 0:28	Python File	1 KB
pyi-set_version.exe	2017/3/28 0:28	应用程序	73 KB
pyi-set_version-script.py	2017/3/28 0:28	Python File	1 KB
pywin32_postinstall.py	2017/3/28 0:27	Python File	25 KB
pywin32_testall.py	2017/3/28 0:27	Python File	4 KB

图9-2

### 3. PyInstaller

PyInstaller 是一个能够将 Python 脚本打包成可执行文件的工具。在 Scripts 文件夹中，可以看到 pyinstaller.exe 文件。

pyinstaller [opts]yourprogram.py

其中：

- -F, -onefile 生成一个可执行文件
- -D, -onedir 生成一个目录，包含可执行文件和资源文件
- -c, -console, -nowindowed 生成控制台应用程序
- -w, -windowed, -noconsole 生成窗口应用程序

### 4. 使用

## PyQt5/Chapter09/example/colorDialog.py

```
from PyQt5.QtWidgets import QApplication, QPushButton, QColorDialog ,
QWidget
from PyQt5.QtCore import Qt
from PyQt5.QtGui import QColor
import sys

class ColorDialog ( QWidget):
    def __init__(self ):
        super().__init__()
        color = QColor(0, 0, 0)
        self.setGeometry(300, 300, 350, 280)
        self.setWindowTitle('颜色选择')
        self.button = QPushButton('Dialog', self)
        self.button.setFocusPolicy(Qt.NoFocus)
        self.button.move(20, 20)
        self.button.clicked.connect(self.showDialog)
        self.setFocus()
        self.widget = QWidget(self)
        self.widget.setStyleSheet('QWidget{background-color:%s}
'%color.name())
        self.widget.setGeometry(130, 22, 100, 100)

    def showDialog(self):
        col = QColorDialog.getColor()
        if col.isValid():

            self.widget.setStyleSheet('QWidget
{background-color:%s}'%col.name())

if __name__ == "__main__":
    app = QApplication(sys.argv)
    qb = ColorDialog()
    qb.show()
    sys.exit(app.exec_())
```

将 colorDialog.py 文件.py 文件放入Python解释器  
中.py文件放入Python 3.5解释器中，运行 CPython解释器  
中C解释器中运行 Python解释器CPython解释器。将colorDialog.py  
文件放入9-3中。

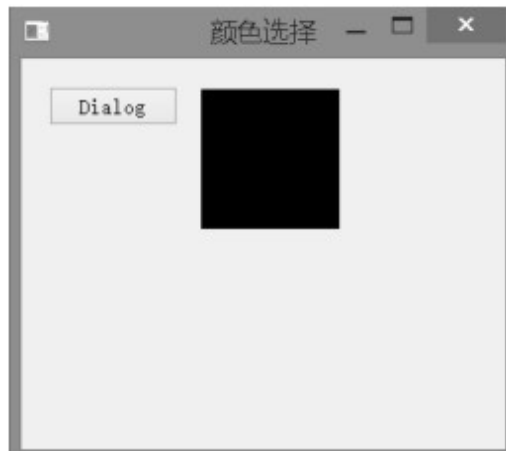


图9-3

将colorDialog.py文件放入Python解释器  
pyinstaller-F-w colorDialog.py  
PyInstaller解释器中运行EXE文件9-4

```
C:\windows\system32\cmd.exe
4698 INFO: running Analysis out00-Analysis.toc
4825 INFO: Caching module hooks...
4843 INFO: Analyzing C:\Users\wangshuo\Desktop\python\colorDialog.py
4861 INFO: Loading module hooks...
4862 INFO: Loading module hook "hook-PyQt5.QtCore.py"...
4956 INFO: Loading module hook "hook-distutils.py"...
4958 INFO: Loading module hook "hook-pydoc.py"...
4959 INFO: Loading module hook "hook-PyQt5.py"...
4962 INFO: Loading module hook "hook-xml.py"...
5326 INFO: Loading module hook "hook-PyQt5.Qt.py"...
5329 INFO: Loading module hook "hook-PyQt5.QtWidgets.py"...
5331 INFO: Loading module hook "hook-encodings.py"...
5487 INFO: Loading module hook "hook-PyQt5.QtGui.py"...
6106 INFO: Loading module hook "hook-PyQt5.QtPrintSupport.py"...
6241 INFO: Looking for ctypes DLLs
6241 INFO: Analyzing run-time hooks ...
6249 INFO: Including run-time hook 'pyi_rth_qt5.py'
6252 INFO: Including run-time hook 'pyi_rth_qt5plugins.py'
6267 INFO: Looking for dynamic libraries
7936 INFO: Looking for eggs
7937 INFO: Using Python library C:\windows\system32\python34.dll
7937 INFO: Found binding redirects:
[]
7947 INFO: Warnings written to C:\Users\wangshuo\Desktop\python\build\colorDialog\warncolorDialog.txt
8006 INFO: checking PYZ
8007 INFO: Building PYZ because out00-PYZ.toc is non existent
8008 INFO: Building PYZ (ZlibArchive) C:\Users\wangshuo\Desktop\python\build\colorDialog\out00-PYZ.pyz
9411 INFO: Building PYZ (ZlibArchive) C:\Users\wangshuo\Desktop\python\build\colorDialog\out00-PYZ.pyz completed successfully.
9445 INFO: checking PKG
9446 INFO: Building PKG because out00-PKG.toc is non existent
9446 INFO: Building PKG (CArchive) out00-PKG.pkg
27792 INFO: Building PKG (CArchive) out00-PKG.pkg completed successfully.
27818 INFO: Bootloader D:\installed_software\Python34\lib\site-packages\PyInstaller\bootloader\Windows-64bit\runw.exe
27819 INFO: checking EXE
27819 INFO: Building EXE because out00-EXE.toc is non existent
27820 INFO: Building EXE from out00-EXE.toc
27821 INFO: Appending archive to EXE C:\Users\wangshuo\Desktop\python\dist\colorDialog.exe
27894 INFO: Building EXE from out00-EXE.toc completed successfully.
```

图9-4

将dist 下的colorDialog.exe复制到 9-5和9-6

图



图9-5



图9-6

colorDialog.exe 是 Python 编译生成的可执行文件，由 colorDialog.py 编译生成。

在

calculator.exe 是 Python 编译生成的可执行文件，由 calculator.py 编译生成。64 位 Python 编译生成的可执行文件是 64 位 Windows 编译生成的，32 位 Python 编译生成的可执行文件是 32 位 Windows 编译生成的。64 位 Windows 编译生成的可执行文件是 64 位 Windows 编译生成的，32 位 Python 编译生成的可执行文件是 32 位 Windows 编译生成的。

## 5. PyInstaller 简介

PyInstaller 是一个 Python 编译生成可执行文件的工具。它可以将 Python 脚本编译生成可执行文件，也可以将 Python 脚本编译生成可执行文件。PyInstaller 支持 Windows、Linux 和 macOS 平台。binutil 是 PyInstaller 的编译工具，用于编译 Python 脚本生成可执行文件。

PyInstaller 是一個跨平台的靜態編譯器，它將 Python 腳本和依賴的庫文件打包成一個可執行的可執行文件。

PyInstaller 是一個跨平台的靜態編譯器，它將 Python 腳本和依賴的庫文件打包成一個可執行的可執行文件。

## 9.2 數據庫

### 9.2.1 SQLite

#### 1. 什麼是 SQLite

SQLite 是一個輕量級的、自給自足的、跨平台的 SQL 數據庫引擎。它是一個嵌入式數據庫，不需要單獨的服務器。

SQLite 是一個輕量級的、自給自足的、跨平台的 SQL 數據庫引擎。

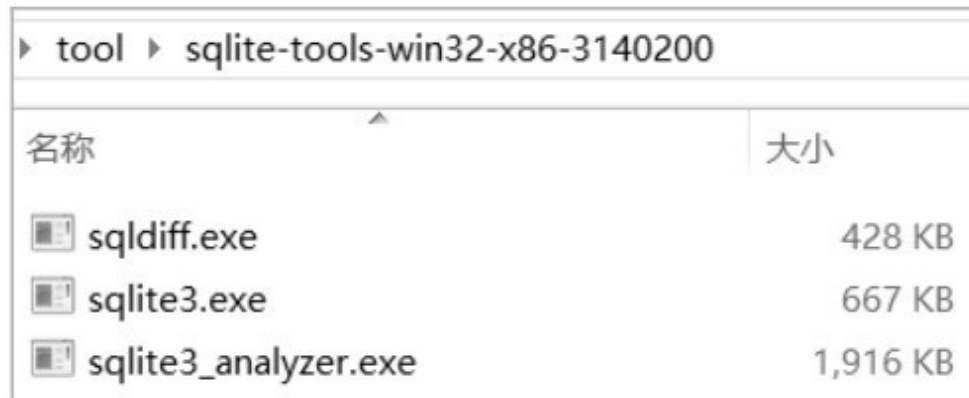
SQLite 是一個輕量級的、自給自足的、跨平台的 SQL 數據庫引擎。

SQLite 實現了 ACID 特性，並支持多種操作系統。它是由 D. Richard Hipp 開發的，目前由 SQLite.org 維護。SQLite 是一個輕量級的、自給自足的、跨平台的 SQL 數據庫引擎。它支持多種操作系統，包括 Windows、Linux、UNIX、Tcl、C#、PHP、Java 等。它還支持 ODBC、MySQL、PostgreSQL 等數據庫的接口。

#### 2. 如何安裝 SQLite

SQLite 的官方下載地址是 <http://www.sqlite.org/download.html>。目前最新的版本是 3.18.0。

sqlite-tools-win32-x86-3140200.zip压缩包解压到任意目录  
sqlite3.exe文件如图9-7所示






tool > sqlite-tools-win32-x86-3140200	
名称	大小
 sqldiff.exe	428 KB
 sqlite3.exe	667 KB
 sqlite3_analyzer.exe	1,916 KB

图9-7

安装SQLite 数据库 Path 环境变量配置如下：SQLite安装  
“环境变量”->“系统”->“环境变量”->“系统”->“Path”->“系统变量”  
Path 环境变量 SQLite 数据库安装路径为：E:\installed\_software\sqlite图9-8所示



□9-8

# sqlite3 9-9 SQLite

```
E:\>sqlite3
SQLite version 3.14.2 2016-09-12 18:50:49
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> _
```

9-9

### 3.SQLite

1

```
cd sqlite3
sqlite3 DatabaseName.db
```

```
testDb.db
```

```
E:\tool\sqlite3 testDb.db
```

```
SQLite version 3.14.2 2016-09-12 18:50:49
```

```
Enter ".help" for usage hints.
```

```
testDB.db SQLite
```

```
sqlite3 "sqlite"
```

2

```
SQLite .databases
```

```
sqlite .databases
```

```
seq name file
```

```
0 main E:\tool\testDb.db
```

```
sqlite
```

3

```
cd sqlite3
```

```
sqlite3 testDb.db
```

testDb.db

E:\tool\sqlite3 testDb.db

SQLite version 3.14.2 2016-09-12 18:50:49

Enter ".help" for usage hints.

sqlite

4

SQLite.help SQLite

sqlite.help

5

SQLite SQL

sqlite>create table people(id integer primary key,name text);

SQLite .db testDb.db

SQLite Windows Linux

Mac OS SQLite sqlite3

testDb.db

SQLite

sqlite> insert into people(id,name) values(1,'zhangsan');

sqlite>insert into people(id,name) values(2,'lisi');

sqlite>insert into people(id,name) values(3,'wangwu');

people SQL

sqlite> select \* from people;

1|zhangsan

2|lisi

3|wangwu

sqlite>people>column>.header on

```
sqlite> .header on
sqlite> select * from people;
id|name
1|zhangsan
2|lisi
3|wangwu
sqlite>
```

sqlite>people>

```
sqlite> .schema people
CREATE TABLE people(id integer primary key,name
text);
sqlite>
```

#### 4. SQLite

Shell SQLite

DBMS SQLite

SQLiteStudio

<https://sqlitestudio.pl/index.rvt>

sqlitestudio-3.1.1.zip

SQLiteStudio.exe SQLiteStudio

SQLiteStudio9-10

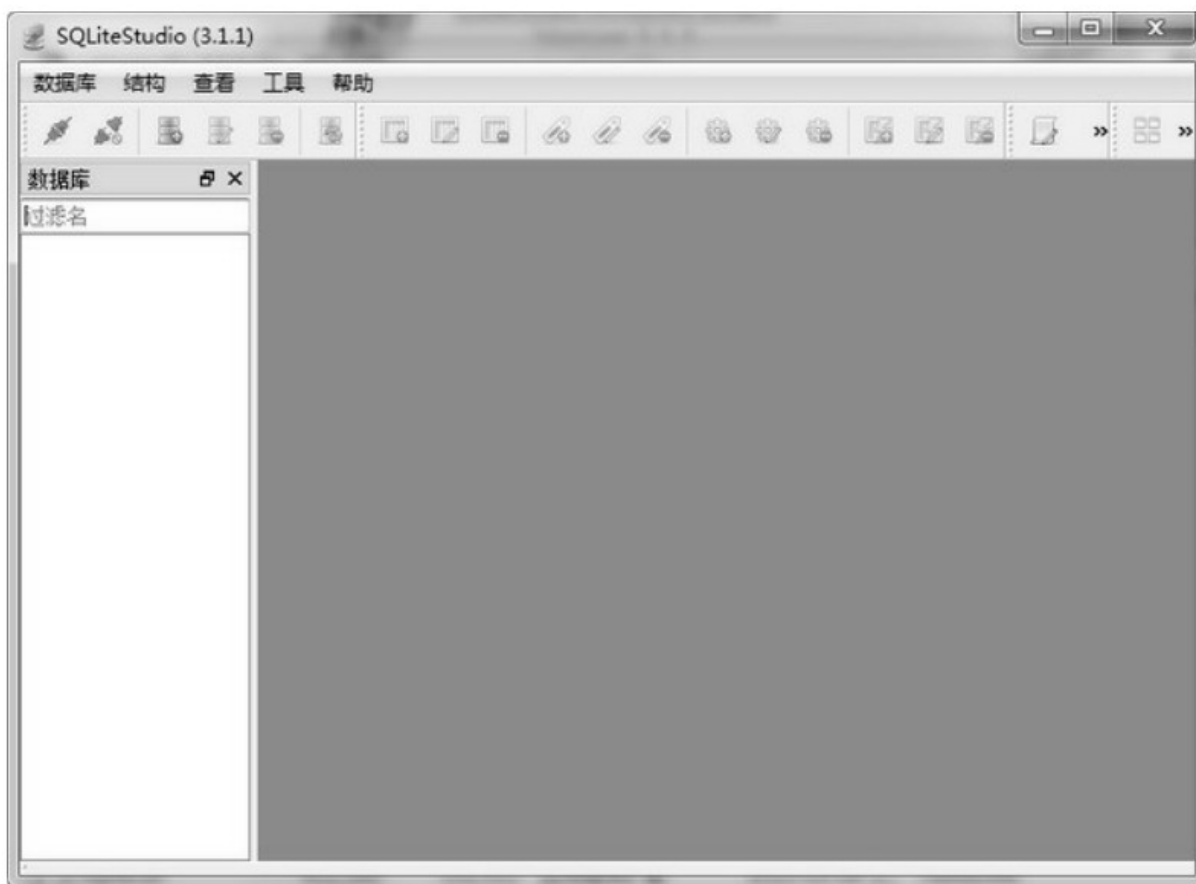


图9-10

SQLiteStudio 支持 SQLite、MySQL、Microsoft SQL Server、Oracle、PostgreSQL、SQLiteStudio 支持将“数据库”→“表”的数据库名 .db 文件保存到本地。

## 9.2.2 数据库

PyQt API 支持 SQLite、MySQL、Microsoft SQL Server、Oracle、PostgreSQL、SQLiteStudio 支持将“数据库”→“表”的数据库名 .db 文件保存到本地。

图9-2

图9-2

数据库驱动类型	描 述
QDB2	IBM DB2 驱动程序
QIBASE	Borland InterBase 驱动程序
QMYSQL	MySQL 驱动程序
QOCI	Oracle 调用接口驱动程序
QODBC	ODBC 驱动程序（包括 Microsoft SQL Server）
QPSQL	PostgreSQL 驱动程序
QSQLITE	SQLite3 或更高版本的驱动程序
QSQLITE2	SQLite2 驱动程序

## QSqlDatabase数据库驱动9-3

9-3

方 法	描 述
addDatabase()	设置连接数据库的数据库驱动类型
setDatabaseName()	设置所连接的数据库名称
setHostName()	设置安装数据库的主机名称
setUserName()	指定连接的用户名
setPassword()	设置连接对象的密码（如果有）
commit()	提交事务，如果执行成功则返回 True
rollback()	回滚数据库事务
close()	关闭数据库连接

addDatabase()数据库驱动类型 QSqlDatabase  
host IP

QSqlDatabase  
addDatabase()数据库驱动类型

MySQL

```
from PyQt5.QtSql import QSqlDatabase
db=QSqlDatabase.addDatabase("QMYSQL")
db.setHostName("192.168.55.110")
db.setDatabaseName("user")
db.setUserName("root")
db.setPassword("ctsi123")
```

```

dbConn=db.open()
#####SQLite#####
from PyQt5.QtSql import QSqlDatabase
db=QSqlDatabase.addDatabase('QSQLITE')
db.setDatabaseName('./db/sports.db')
# #####
dbConn=db.open()

```

### 9.2.3 SQL

```

QSqlQuery#####SQL#####DDL DML#####SQL
##### exec_()##### SQL #####
query=QSqlQuery()
query.exec_("create table people(id int primary
key,name varchar(20),address varchar(30))")
##### PyQt5/Chapter09/qt09_db01.py##### SQLite
#####database.db#####people#####5#####

```

```

import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from PyQt5.QtSql import QSqlDatabase , QSqlQuery

def createDB():
    db = QSqlDatabase.addDatabase('QSQLITE')
    db.setDatabaseName('./db/database.db')

    if not db.open():
        QMessageBox.critical(None, ("无法打开数据库"),
            ( "无法建立到数据库的连接,这个例子需要 SQLite 支持, 请检查数据库配置。
\n\n 单击取消按钮退出应用。"),
            QMessageBox.Cancel )
        return False

    query = QSqlQuery()
    query.exec_("create table people(id int primary key, name
varchar(20), address varchar(30))")
    query.exec_("insert into people values(1, 'zhangsan1', 'BeiJing')")
    query.exec_("insert into people values(2, 'lisi1', 'TianJing')")
    query.exec_("insert into people values(3, 'wangwu1', 'HenNan')")
    query.exec_("insert into people values(4, 'lisi2', 'HeBei')")
    query.exec_("insert into people values(5, 'wangwu2', 'shanghai')")
    # 关闭数据库
    db.close()
    return True

if __name__ == '__main__':
    app = QApplication(sys.argv)
    createDB()
    sys.exit(app.exec_())

```

使用SQLiteStudio打开database.db文件，可以看到已经创建好了5条数据





图9-11

在SQL语句执行完毕后，使用`db.close()`关闭数据库连接。在Python中，使用`SQLAlchemy`库创建数据库连接对象，使用`db.close()`关闭数据库连接。在Python中，使用`SQLAlchemy`库创建数据库连接对象，使用`db.close()`关闭数据库连接。

在PyQt5中，使用`PyQt5.QtWidgets.QApplication`类创建应用程序对象。在PyQt5中，使用`PyQt5.QtWidgets.QApplication`类创建应用程序对象。在PyQt5中，使用`PyQt5.QtWidgets.QApplication`类创建应用程序对象。

```

class ExecDatabaseDemo(QWidget):

    def __init__(self, parent=None):
        super(ExecDatabaseDemo, self).__init__(parent)

        self.db = QSqlDatabase.addDatabase('QSQLITE')
        self.db.setDatabaseName('./db/database.db')
        # 打开数据库
        self.db.open()

    def closeEvent(self, event):
        # 关闭数据库
        self.db.close()

```

```

if __name__ == '__main__':
    app = QApplication(sys.argv)
    demo = ExecDatabaseDemo()
    demo.show()
    sys.exit(app.exec_())

```

## 9.2.4 数据库

PyQt 的 QSqlTableModel 类用于处理数据库模型，QTableView 类用于显示数据库模型。QSqlTableModel 类是 QSqlDatabase 类的子类，它提供了对数据库表的访问。QTableView 类是 QWidget 类的子类，它提供了对数据库模型的视图。

QSqlTableModel 类提供了以下方法：

```

model=QtSql.QSqlTableModel()
model.setTable("people")
model.setEditStrategy(QSqlTableModel.OnManualSubmit)
model.setFilter("id > 1")
model.select()

```

```

model.setHeaderData(0,Qt.Horizontal,"id")
model.setHeaderData(1,Qt.Horizontal,"name")
model.setHeaderData(2,Qt.Horizontal,"address")
view=QTableView(self)
view.setModel(model)
view.show()

```

QSqlTableModel 的 setTable() 方法用于设置要操作的数据库表名。setFilter() 方法用于设置 SQL 查询语句，其中 where 子句用于过滤数据，select() 方法用于设置要显示的列。setEditStrategy() 方法用于设置编辑策略。

图 9-4

编辑策略	描 述
QSqlTableModel.OnFieldChange	所有变更实时更新到数据库中
QSqlTableModel.OnRowChange	当用户选择不同的行时，在当前行进行变更
QSqlTableModel.OnManualSubmit	手动提交，不自动提交

在 PyQt5/Chapter09/qt09\_db03.py 文件中，PyQt 5 的 QSqlTableModel 类用于操作数据库。

```

import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from PyQt5.QtSql import QSqlDatabase , QSqlTableModel
from PyQt5.QtCore import Qt

def initializeModel(model):
    model.setTable('people')
    model.setEditStrategy( QSqlTableModel.OnFieldChange)
    model.select()
    model.setHeaderData(0, Qt.Horizontal, "ID")
    model.setHeaderData(1, Qt.Horizontal, "name")
    model.setHeaderData(2, Qt.Horizontal, "address")

def createView(title, model):
    view = QTableView()
    view.setModel(model)
    view.setWindowTitle(title)
    return view

def addrow():
    ret = model.insertRows( model.rowCount(), 1 )
    print( 'insertRows=%s' %str(ret) )

def findrow(i):
    delrow= i.row()
    print('del row=%s' % str(delrow) )

if __name__ == '__main__':
    app = QApplication(sys.argv)
    db = QSqlDatabase.addDatabase('QSQLITE')
    db.setDatabaseName('./db/database.db')
    model = QSqlTableModel()
    delrow = -1
    initializeModel(model)
    view1 = createView("Table Model (View 1)", model)
    view1.clicked.connect(findrow)

    dlg= QDialog()
    layout = QVBoxLayout()
    layout.addWidget(view1)

```

```

addBtn = QPushButton("添加一行")
addBtn.clicked.connect(addrow)
layout.addWidget(addBtn)

delBtn = QPushButton("删除一行")
delBtn.clicked.connect(lambda:
    model.removeRow(view1.currentIndex().row()))
layout.addWidget(delBtn)
dlg.setLayout(layout)
dlg.setWindowTitle("Database 例子")
dlg.resize(430, 450)
dlg.show()
sys.exit(app.exec_())

```

□□□□□□□□□□9-12□□□



```

class PeopleTableModel(QSqlTableModel):
    def __init__(self, parent=None, db=None, table='people'):
        super(PeopleTableModel, self).__init__(parent, db, table)
        self.setEditStrategy(QSqlTableModel.OnFieldChange)
        self.select()

class PeopleTableView(QTableView):
    def createView(title, model):
        view = QTableView()
        view.setModel(model)
        view.setWindowTitle(title)
        return view

class PeopleDialog(QDialog):
    def __init__(self, parent=None):
        super(PeopleDialog, self).__init__(parent)
        self.addBtn.clicked.connect(self.addrow)

    def addrow(self):
        ret = model.insertRows(model.rowCount(), 1)
        print('insertRows=%s' % str(ret))

```

9-13 添加新记录

“ 添加 ” 按钮点击事件连接到 addrow() 方法，该方法调用 QSqlTableModel 的 insertRows() 方法添加新记录。



图9-13

```

// 删除一行
delBtn=QPushButton("删除一行")
delBtn.clicked.connect(lambda:
model.removeRow(view1.currentIndex().row()))

```

## 9.2.5 数据库操作

PyQt5/Chapter09/DataGrid.py  
9-14



图9-14

1. 数据库

SQLite数据库student SQLite数据库 database.db PyQt5/Chapter09/db student 9-5

图9-5

列 名	数据类型	名 称	规 则
id	Int	编号，记录的唯一标识符，不能重复	主键（Primary Key）
name	Varchar	名字	
sex	Varchar	性别	
age	Int	年龄	
deparment	Varchar	系	

student SQLite Python

SQLitestudent SQLitestudent



E:\quant\PyQt5\Chapter05\db \sqlite3 datagrid.db

SQLite version 3.14.2 2016-09-12 18:50:49

Enter ".help" for usage hints.

```
sqlite> create table student(id int primary key,name
vchar,sex vchar,age int,deparment vchar) ;
```

```
sqlite> student>10>
```

```
sqlite>insert into student values(1,'001','M',20,'000') ;
```

```
sqlite>insert into student values(2,'001','M',19,'000') ;
```

```
sqlite>insert into student values(3,'001','M',22,'000') ;
```

```
sqlite>insert into student values(4,'001','M',21,'000') ;
```

```
sqlite>insert into student values(5,'001','M',20,'000') ;
```

```
sqlite>insert into student values(6,'001','M',19,'000') ;
```

```
sqlite>insert into student values(7,'001','M',20,'000') ;
```

```
sqlite>insert into student values(8,'001','M',19,'000') ;
```

```
sqlite>insert into student values(9,'002','M',21,'000') ;
```

```
sqlite>insert into student values(10,'003','M',20,'000') ;
```

```
sqlite> student>10>
```

```
sqlite>Python>student>
```

```
sqlite>
```

```

def createTableAndInit():
    # 添加数据库
    db = QSqlDatabase.addDatabase('QSQLITE')
    # 设置数据库名称
    db.setDatabaseName('./db/database.db')
    # 判断是否打开数据库
    if not db.open():
        return False

    # 声明数据库查询对象
    query = QSqlQuery()
    # 创建表
    query.exec("create table student(id int primary key, name vchar, sex
vchar, age int, deparment vchar)")

    # 添加记录
    query.exec("insert into student values(1,'张三 1','男',20,'计算机')")
    query.exec("insert into student values(2,'李四 1','男',19,'经管')")
    query.exec("insert into student values(3,'王五 1','男',22,'机械')")
    query.exec("insert into student values(4,'赵六 1','男',21,'法律')")
    query.exec("insert into student values(5,'小明 1','男',20,'英语')")
    query.exec("insert into student values(6,'小李 1','女',19,'计算机')")
    query.exec("insert into student values(7,'小张 1','男',20,'机械')")
    query.exec("insert into student values(8,'小刚 1','男',19,'经管')")
    query.exec("insert into student values(9,'张三 2','男',21,'计算机')")
    query.exec("insert into student values(10,'张三 3','女',20,'法律')")

    return True

```

## 2. 数据库操作

在 `__init__()` 方法中，我们使用 `QSqlDatabase.addDatabase()` 方法添加数据库，并设置数据库名称。然后，我们使用 `QSqlDatabase.open()` 方法打开数据库。如果打开失败，我们返回 `False`。否则，我们创建数据库查询对象 `QSqlQuery`，并执行 SQL 语句来创建表。最后，我们添加 10 条记录。

文件 `PyQt5/Chapter09/DataGrid.py`

```

class DataGrid(QWidget):

    def __init__(self):
        super().__init__()
        self.setWindowTitle("分页查询例子")
        self.resize(750,300)

        # 查询模型
        self.queryModel = None
        # 数据表
        self.tableView = None
        # 总页数文本
        self.totalPageLabel = None
        # 当前页文本
        self.currentPageLabel = None
        # 转到页输入框
        self.switchPageLineEdit = None
        # 前一页按钮
        self.prevButton = None
        # 后一页按钮
        self.nextButton = None
        # 转到页按钮
        self.switchPageButton = None
        # 当前页
        self.currentPage = 0
        # 总页数
        self.totalPage = 0
        # 总记录数
        self.totalRecrodCount = 0
        # 每页显示记录数
        self.PageRecordCount = 5

```

```

# 初始化
operatorLayout=QHBoxLayout()

```

```

self.prevButton=QPushButton("⏮")
self.nextButton=QPushButton("⏭")
self.switchPageButton=QPushButton("Go")
self.switchPageLineEdit=QLineEdit()
self.switchPageLineEdit.setFixedWidth(40)
switchPage=QLabel("000")
page=QLabel("")
operatorLayout.addWidget(self.prevButton)
operatorLayout.addWidget(self.nextButton)
operatorLayout.addWidget(switchPage)
operatorLayout.addWidget(self.switchPageLineEdit)
operatorLayout.addWidget(page)
operatorLayout.addWidget(self.switchPageButton)
operatorLayout.addWidget( QSplitter())
#####
###
    # #####
    self.tableView=QTableView()
    self.tableView.horizontalHeader().setStretchLastSection
(True)
    self.tableView.horizontalHeader().setSectionResizeMod
e(
        QHeaderView.Stretch)
#####
    mainLayout=QVBoxLayout(self);
    mainLayout.addLayout(operatorLayout);
    mainLayout.addWidget(self.tableView);

```

```
mainLayout.addLayout(statusLayout);  
self.setLayout(mainLayout)
```

图9-15



图9-15

### 3. 数据库操作

使用 QSqlQueryModel 类操作 database.db 数据库中的 student 表。  
QSqlQueryModel 类的使用方法如下：



```
self.updateStatus()

# 后一页按钮被按下
def onNextButtonClick(self):
    print('*** onNextButtonClick ');
    limitIndex = self.currentPage * self.PageRecordCount
    self.recordQuery( limitIndex)
    self.currentPage += 1
    self.updateStatus()

# 跳转按钮被按下
def onSwitchPageButtonClick(self):
    # 得到输入字符串
    szText = self.switchPageLineEdit.text()
    # 数字正则表达式
    pattern = re.compile(r'^[-+]?[0-9]+\.[0-9]+$')
    match = pattern.match(szText)

    # 判断是否为数字
    if not match :
        QMessageBox.information(self, "提示", "请输入数字" )
        return

    # 是否为空
    if szText == '' :
        QMessageBox.information(self, "提示" , "请输入跳转页面" )
        return

    # 得到页数
    pageIndex = int(szText)
    # 判断是否有指定页
    if pageIndex > self.totalPage or pageIndex < 1 :
        QMessageBox.information(self, "提示",
            "没有指定的页面，请重新输入" )
        return

    # 得到查询起始行号
    limitIndex = (pageIndex-1) * self.PageRecordCount

    # 记录查询
    self.recordQuery(limitIndex);
```

## 9.3 Pandas与PyQt的结合

Pandas与Python的结合始于AQR Capital Management在2008年4月发布的2009年发布的Python与PyData的结合。PyData与Pandas的结合始于Panel Data与Python的Data Analysis的结合。Pandas与Python的结合始于NumPy与Pandas的结合。Python与Pandas的结合始于Pandas与PyQt的结合。

与Pandas与PyQt的结合始于qtpandas与Pandas的结合。Pandas与QTableWidget的结合始于QTableWidget与Pandas的结合。Pandas与QTableWidget的结合始于qtpandas与Pandas的结合。

### 9.3.1 qtpandas

安装Pandas与pip  
pip install pandas  
pip 安装Pandas 与TUNA  
TUNA与pypi 5  
TUNA与simple  
https与http

pip install-i https://pypi.tuna.tsinghua.edu.cn/simple pandas

安装qtpandas与Pandas与pip

pip install qtpandas

pip install qtpandas 1.03  
1.03与qtpandas与PyQt4



qtpandas需要PyQt5

qtpandas1.04需要PyQt 5  
git pandas

gitqtpandas

gitqtpandas

git clone https://github.com/draperjames/qtpandas.git

cd qtpandas

python setup.py install

qtpandas

qtpandas

https://github.com/draperjames/qtpandas

Code “Clone or download”  
“Download ZIP” qtpandas qtpandas-master.zip  
zipqtpandas

cd qtpandas

python setup.py install

pip show qtpandas

C:\Users\si\Downloads\qtpandas-master pip show qtpandas

Name: qtpandas

Version: 1.0.4

Summary: Utilities to use pandas (the data analysis / manipulation

Home-page: None

Author: None

Author-email: None

License: None

Location: e:\installed\_software\python35\lib\site-packages\  
qtpandas-1.0.4-py3.5.

Requires: pandas,easygui,pytest,pytest-qt,qtpy,future,pytest-cov

qtpandas1.0.4

qtpandasPythonimport  
qtpandas

### 9.3.2

```
from __future__ import unicode_literals  
from __future__ import print_function
```

```
from __future__ import division
from __future__ import absolute_import
from future import standard_library
standard_library.install_aliases()
import pandas
import numpy
import sys
from qtpandas.excepthook import excepthook

# 使用 compat 模块中的 QtGui 类，请确保安装了必需的 sip 库
from qtpandas.compat import QtGui
from qtpandas.models.DataFrameModel import DataFrameModel
from qtpandas.views.DataTableView import DataTableWidget
# from qtpandas.views._ui import icons_rc

sys.excepthook = excepthook # 设置 PyQt 的异常钩子，在本例中基本没什么用

# 创建一个空的模型，该模型用于存储与处理数据
model = DataFrameModel()

# 创建一个应用用于显示表格
app = QtGui.QApplication([])
widget = DataTableWidget() # 创建一个空的表格，主要用来呈现数据
widget.resize(500, 300) # 调整 Widget 的大小
widget.show()
# 让表格绑定模型，也就是让表格呈现模型的内容
widget.setViewModel(model)

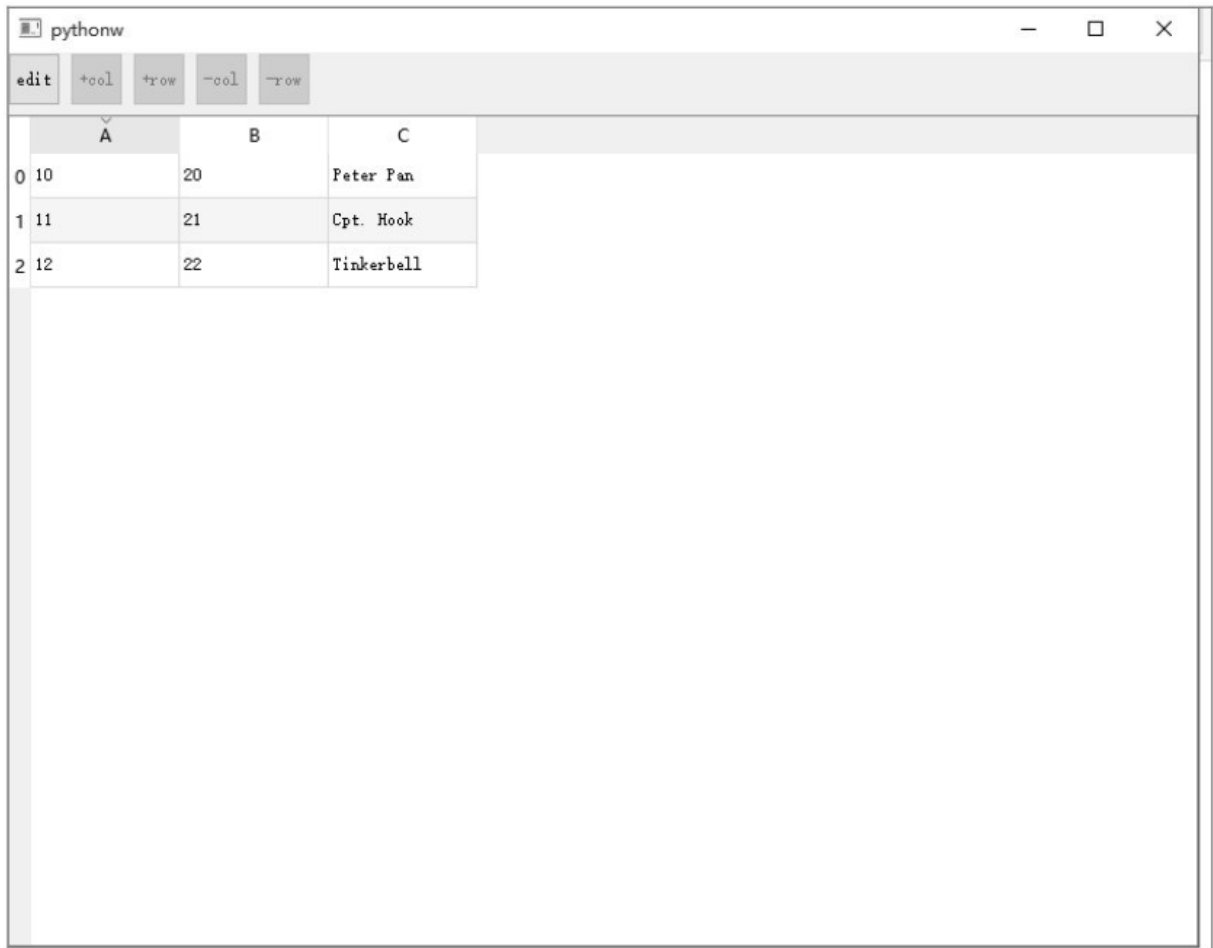
# 创建测试数据
data = {
    'A': [10, 11, 12],
    'B': [20, 21, 22],
    'C': ['Peter Pan', 'Cpt. Hook', 'Tinkerbell']
}
df = pandas.DataFrame(data)

# 下面两列用来测试委托是否成立
df['A'] = df['A'].astype(numpy.int8) # A 列数据格式变成整型
df['B'] = df['B'].astype(numpy.float16) # B 列数据格式变成浮点型

# 在模型中填入数据 df
model.setDataFrame(df)
```

```
# 启动程序
app.exec_()
```

BasicExample.py 的完整代码可以在 <https://github.com/draperjames/qtpandas/tree/master/examples> 中找到 9-16



	A	B	C
0	10	20	Peter Pan
1	11	21	Cpt. Hook
2	12	22	Tinkerbell

图 9-16

图 9-16 展示了使用 Qt 和 pandas 库创建的一个简单应用程序。该应用程序显示了一个包含 3 行 3 列数据的数据框。图 9-17 展示了如何使用 Qt 和 pandas 库来创建和显示数据框。图 9-18 展示了如何使用 Qt 和 pandas 库来创建和显示数据框。图 9-19 展示了如何使用 Qt 和 pandas 库来创建和显示数据框。图 9-20 展示了如何使用 Qt 和 pandas 库来创建和显示数据框。

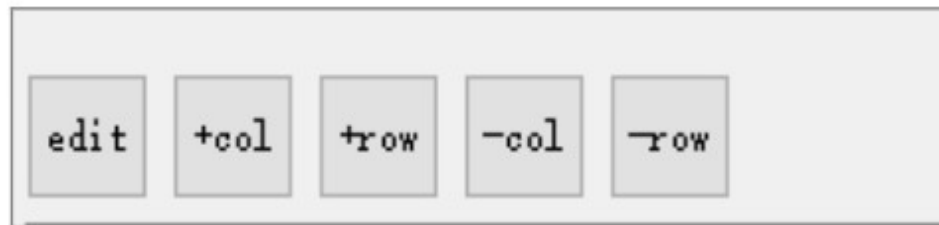


Figure 9-17

	A	B	C
0	10	20	Peter Pan
1	11	21	Cpt. Hook
2	12	22	Tinkerbell

Figure 9-18

	A	B	C
0	10	20	Peter Pan
1	11	21.0	Cpt. Hook
2	12	22	Tinkerbell

□9-19

<div> <div>edit</div> <div>+col</div> <div>+row</div> <div>-col</div> <div>-row</div> </div>			
	A	B	C
0	10	20	Peter Pan
1	11	21	Ept. Hook
2	12	22	Tinkerbell

□9-20

9-21



```
# 顯示數據框df
```

```
model.setDataFrame(df)
```

在 Qt Designer 中，Pandas 的 DataFrame 模型可以通過 Qt Designer 中的 DataTableWidget 來顯示。DataFrameModel 是 Qt Designer 中的一個模型，它可以用來顯示數據框。在 Qt Designer 中，你可以通過拖拽 DataFrameModel 到視圖中來顯示數據框。

### 9.3.3 數據框模型

在 Qt Designer 中，Pandas 的 DataFrame 模型可以通過 Qt Designer 中的 DataTableWidget 來顯示。DataFrameModel 是 Qt Designer 中的一個模型，它可以用來顯示數據框。在 Qt Designer 中，你可以通過拖拽 DataFrameModel 到視圖中來顯示數據框。

Container 是一個 QWidget 的子類，它可以用來顯示數據框。在 Qt Designer 中，你可以通過拖拽 Container 到視圖中來顯示數據框。

“Qt” 是一個 “跨平台” 的軟件庫。9-22 是一個數據框模型。



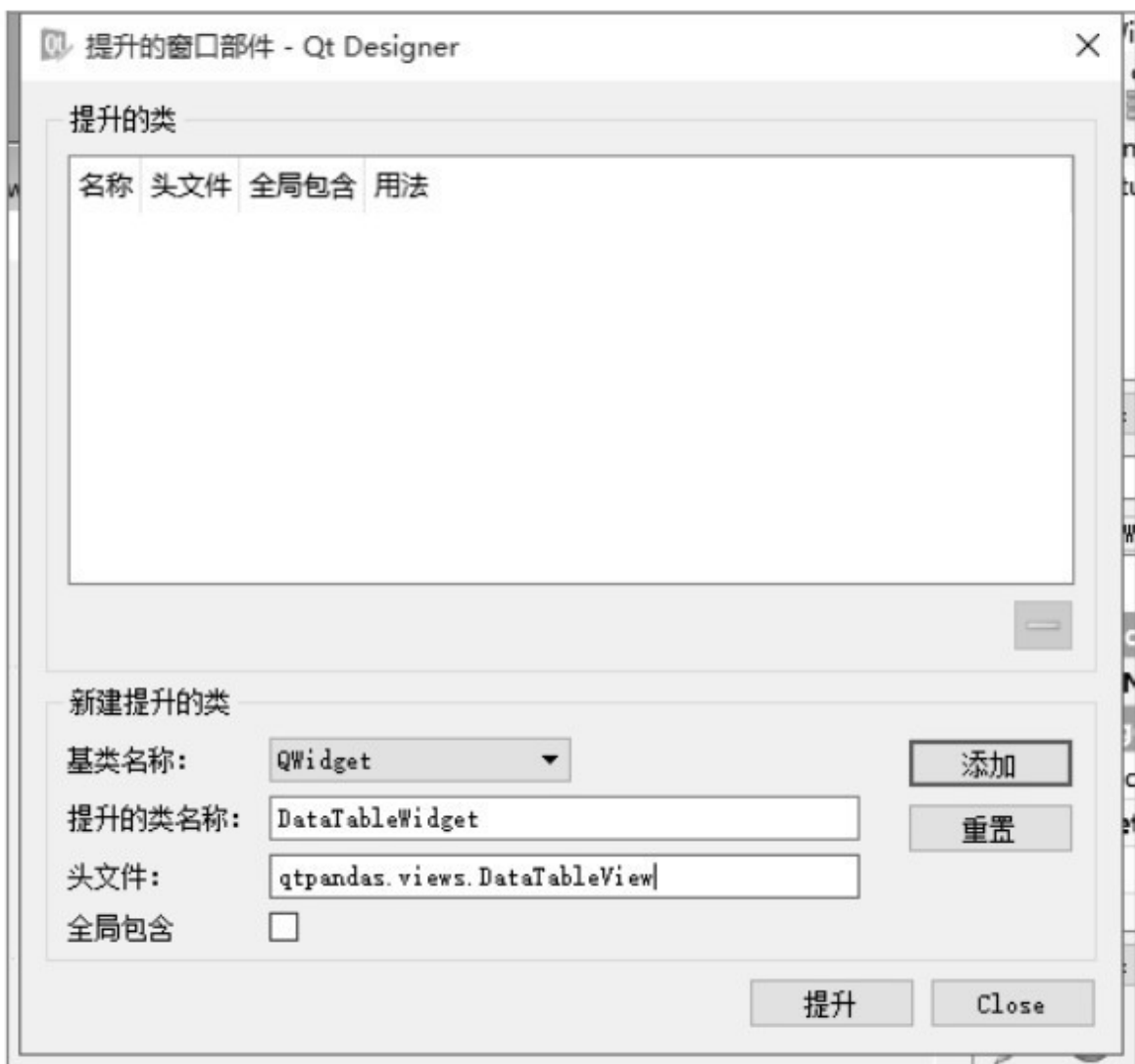


图9-22

在“提升”对话框中，“名称”文本框中输入9-23

“头文件”文本框中输入9-24

Qt Designer中将DataTableWidget

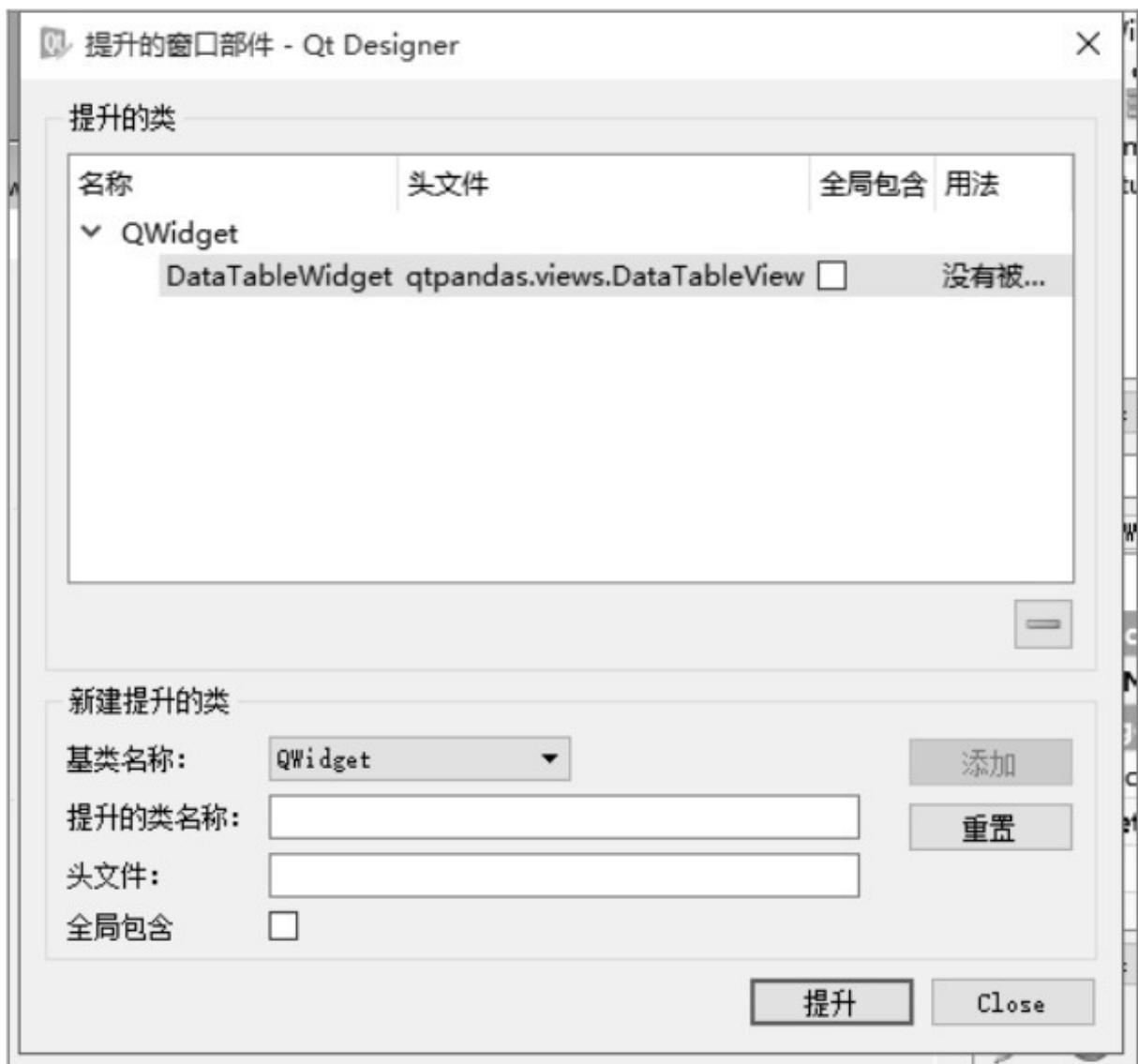


图9-23

对象查看器	
对象	类
▼ MainWindow	QMainWindow
▼ centralwidget	QWidget
widget	DataTableWidget
menubar	QMenuBar
statusbar	QStatusBar

图9-24

在 `Widget` 中添加“`pandastablewidget`”对象并设置其属性。

```
from qtpandas.views.DataTableView import
DataTableWidget

self.pandastablewidget=DataTableWidget(self.centralW
idget)

self.pandastablewidget.setGeometry(QRect(10,
30,591,331))

self.pandastablewidget.setStyleSheet("")

self.pandastablewidget.setObjectName("pandastablewi
dget")
```

在 `DataTableWidget` 中添加 `Qt Designer` 对象。

在 `PyQt` 中添加 `Qt Designer` 对象。

在 `PyQt` 中添加 `Python` 对象。

### [9.3.4 qtpandas](#)

单击事件被捕获并打印到控制台，显示为 clicked 9-25

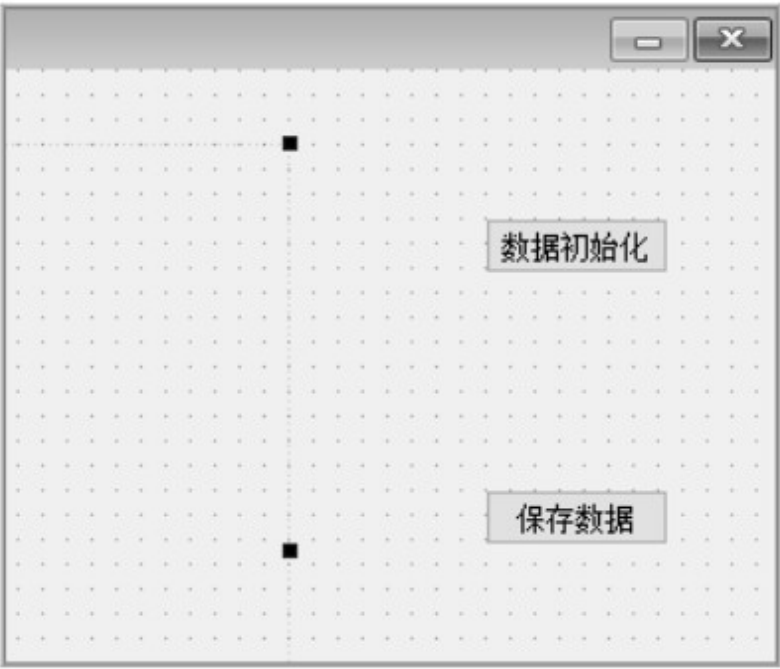


图9-25

单击事件被捕获并打印到控制台，显示为 clicked 9-25

```

# -*- coding: utf-8 -*-

"""
Module implementing MainWindow.
"""

from PyQt5.QtCore import pyqtSlot
from PyQt5.QtWidgets import QMainWindow, QApplication

from Ui_pandas_pyqt import Ui_MainWindow

from qtpandas.models.DataFrameModel import DataFrameModel
import pandas as pd

class MainWindow(QMainWindow, Ui_MainWindow):
    """
        Class documentation goes here.
    """
    def __init__(self, parent=None):
        """
            Constructor

            @param parent reference to the parent widget
            @type QWidget

```

```

"""
super(MainWindow, self).__init__(parent)
self.setupUi(self)

    '''初始化 pandasqt'''
widget = self.pandastablewidget
    widget.resize(600, 500) # 如果对控件尺寸不满意, 可以在这里设置

    self.model = DataFrameModel() # 设置新的模型
widget.setModel(self.model)

    self.df =
pd.read_excel(r'./data/fund_data.xlsx', encoding='gbk')
    self.df_original = self.df.copy() # 备份原始数据
self.model.setDataFrame(self.df)

@pyqtSlot()
def on_pushButton_clicked(self):
    """
    初始化 pandas
    """
    self.model.setDataFrame(self.df_original)

@pyqtSlot()
def on_pushButton_2_clicked(self):
    """
    保存数据
    """
    self.df.to_excel(r'./data/fund_data_new.xlsx')

if __name__ == "__main__":
    import sys

    app = QApplication(sys.argv)
    ui = MainWindow()
    ui.show()
    sys.exit(app.exec_())

```

图9-26

MainWindow

edit+col+row-col-row

	公司名称	基金ID	基金简称	发行时间	基金管理人	净值
0	有容投资	HF000009L2	有容基金	2014/4/1 星期二 0:00	黄卫民	300.392086
1	优素资产	HF00000686	优素片叶1号	2007/1/4 星期四 0:00	李灵活	278.818394
2	固利资产	HF0000115X	固利资产趋势进取策略	2015/3/16 星期一 0:00	王兵	124.00240
3	誉信丰投资	HF000008M1	乐活恒永	2011/8/2 星期二 0:00	陈世彬	71.46144
4	None	HF000009EG	中蕴投资	2012/4/19 星期四 0:00	None	48.508245
5	凯泽投资	HF000004ZD	凯泽一号	2013/5/2 星期四 0:00	毛泽红	41.436033
6	洪门投资	HF000010XM	洪门自营	2015/1/1 星期四 0:00	洪冲	27.18
7	熙然投资	HF00000ECV	熙然1号	2014/2/25 星期二 0:00	None	26.554627
8	莲华投资	HF000009YW	小象计划	2012/7/10 星期二 0:00	祥和	26.332078
9	英恺投资	HF00000BVD	趋势神剑-金山	2012/12/25 星期二 0:00	唐恺玲	25.801153
10	好运来工作室	HF00000F83	好运来一号	2015/1/2 星期五 0:00	None	24.5081
11	鸿凯投资	HF000006QZ	鸿凯激进一号	2013/11/22 星期五 0:00	林军	22.530573
12	奇获投资	HF000008QC	奇获投资开拓型	2014/1/7 星期二 0:00	孟德稳	22.132193
13	紫熙投资	HF00000VIF	紫熙激进一号	2014/5/14 星期二 0:00	严忠	21.484707

数据初始化保存数据

图9-26

图9-27

图9-27



图9-27

图9-28展示了基金数据文件fund\_data\_new.xlsx的初始数据。

A	B	C	D	E
	基金ID	投资策略	子策略	
31	HF000000CC	管理期货	主观套利	
4	HF0000009E	管理期货	系统化趋势	
3	HF0000008M	管理期货	系统化高频	
1	HF000000G8	宏观策略	宏观策略	
34	HF0000010Z	股票策略	股票多头	
5	HF0000004Z	管理期货	主观趋势	
16	HF00000063	管理期货	系统化趋势	
15	HF00000002	股票策略	股票多头	
24	HF00000004	管理期货	主观趋势	

图9-28

图9-29展示了基金数据的初始数据。

## 9.4 Matplotlib与PyQt的结合

本章将介绍如何将Python中的Matplotlib库与PyQt库结合使用，以实现数据可视化。Matplotlib是一个强大的数据可视化库，而PyQt是一个用于创建图形用户界面的库。



Matplotlib Python 的接口，与 MATLAB 的 API 类似，  
可以方便地集成到各种 GUI 中。

Matplotlib 的 Gallery 页面：  
<http://matplotlib.org/gallery.html>  
Matplotlib 与 PyQt 集成  
PyQt 的 Gallery 页面：  
PyQt 与 Matplotlib 集成  
PyQt 的 Gallery 页面：

MatplotlibWidget.py 与 Matplotlib 与 PyQt 集成

### 9.4.1 MatplotlibWidget

#### 1. 简介

PyQt5/Chapter09/MatplotlibWidget.py 与  
FigureCanvas 集成  
Matplotlib

```
class MyMplCanvas(FigureCanvas):
    """FigureCanvas 的最终父类其实是 QWidget"""

    def __init__(self, parent=None, width=5, height=4, dpi=100):

        # 设置中文显示
        plt.rcParams['font.family'] = ['SimHei'] # 用来正常显示中文标签
        plt.rcParams['axes.unicode_minus'] = False # 用来正常显示负号

        # 新建一个绘图对象
```

```

self.fig = Figure(figsize=(width, height), dpi=dpi)
# 建立一个子图。如果要建立复合图，可以在这里修改
self.axes = self.fig.add_subplot(111)

self.axes.hold(False) # 每次绘图时都不保留上一次绘图的结果

FigureCanvas.__init__(self, self.fig)
self.setParent(parent)

'''定义 FigureCanvas 的尺寸策略，意思是设置 FigureCanvas，使之尽可能向
外填充空间'''
FigureCanvas.setSizePolicy(self,
                             QSizePolicy.Expanding,
                             QSizePolicy.Expanding)
FigureCanvas.updateGeometry(self)

```

```

class Gallery:
    def start_static_plot(self):
        self.fig.suptitle('')
        t=arange(0.0,3.0,0.01)
        s=sin(2 * pi * t)
        self.axes.plot(t,s)
        self.axes.set_ylabel('Y')
        self.axes.set_xlabel('X')
        self.axes.grid(True)

    def update_figure(self):

```



```

class MatplotlibWidget(QWidget):
    def __init__(self, parent=None):
        super(MatplotlibWidget, self).__init__(parent)
        self.initUi()

    def initUi(self):
        self.layout = QVBoxLayout(self)
        self.mpl = MyMplCanvas(self, width=5, height=4, dpi=100,
title='Title 1')
        # self.mpl.start_static_plot() # 如果想要在初始化时就呈现静态图，请取消这行注释
        # self.mpl.start_dynamic_plot() # 如果想要在初始化时就呈现动态图，请取消这行注释
        self.mpl_ntb = NavigationToolbar(self.mpl, self) # 添加完整的工具栏

        self.layout.addWidget(self.mpl)
        self.layout.addWidget(self.mpl_ntb)

```

□□□□□

```

if __name__ == '__main__':
    app=QApplication(sys.argv)
    ui=MatplotlibWidget()
    ui.mpl.start_static_plot() # □□□□□□□□
    # ui.mpl.start_dynamic_plot() # □□□□□□□□
    ui.show()
    sys.exit(app.exec_())

```

□□□□9-29□□□□□□□□□□□□□□□□

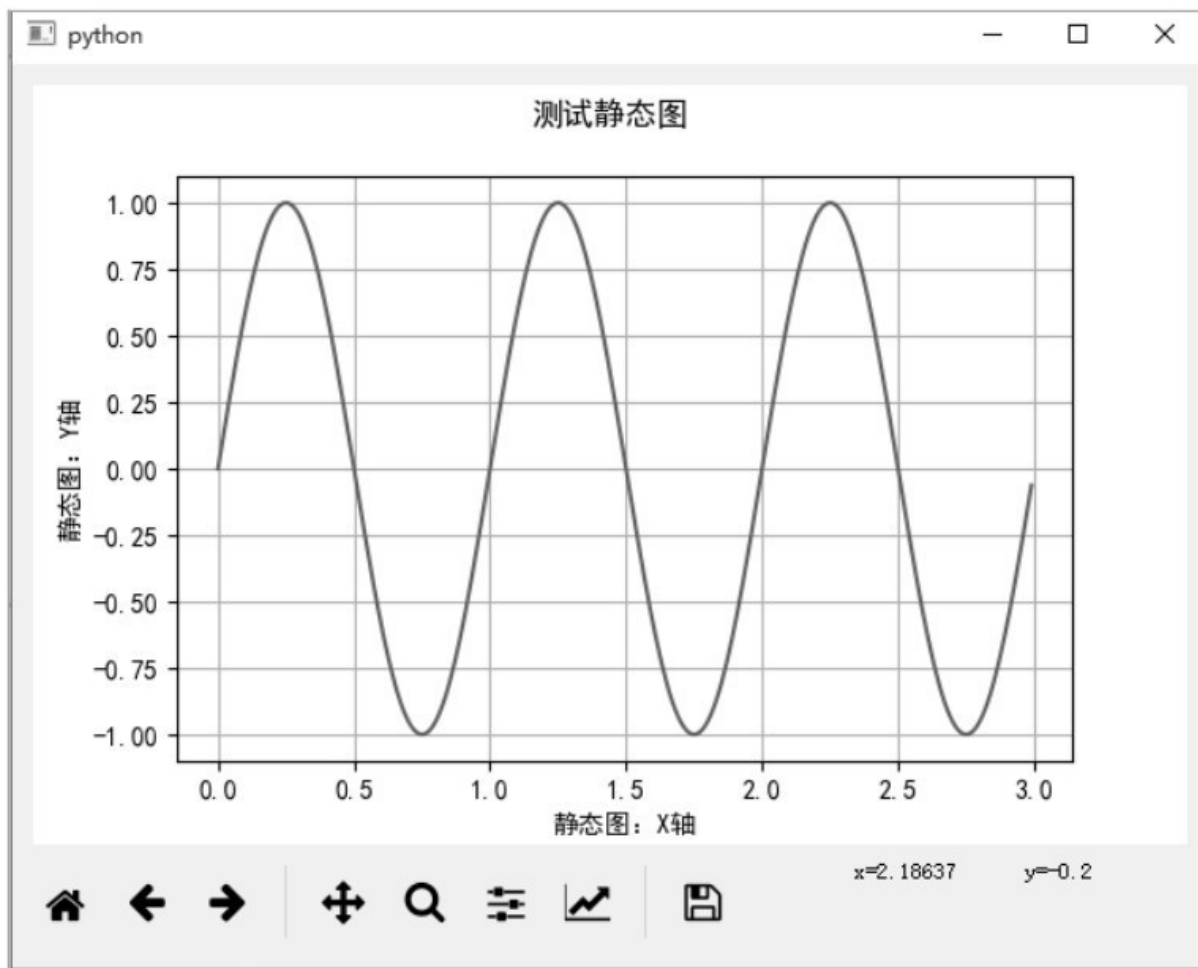


图9-29

## 9.4.2 静态图

在 9.3.3 节中，我们使用 Qt Designer 创建了一个静态图，该图使用 Matplotlib 和 PyQt 库。

在 Qt Designer 中，我们使用 PyQt5/Chapter09/matplotlib\_pyqt.ui 文件。

该文件是一个 QWidget 窗口，如图 9-30 所示。

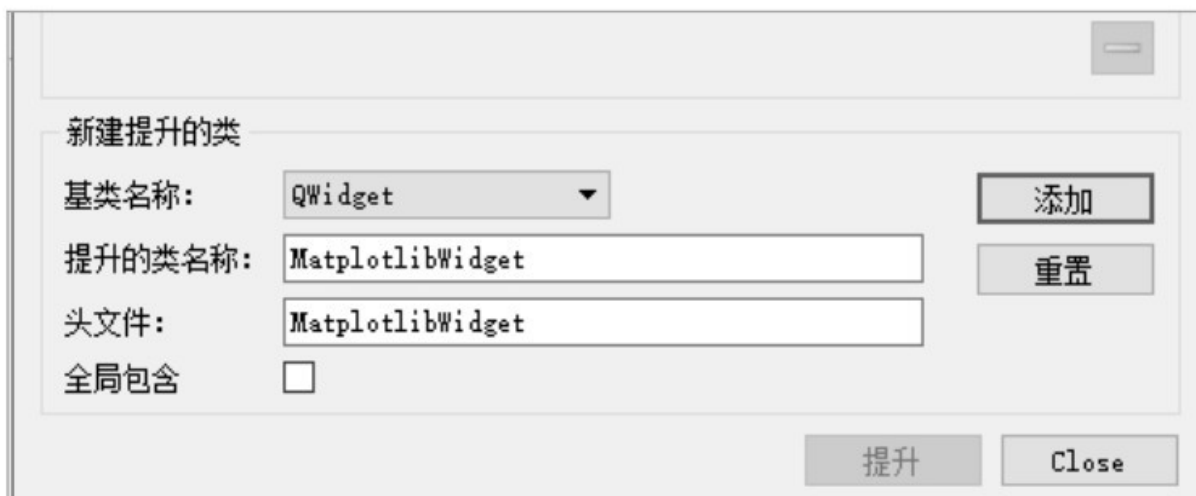


图9-30

图9-31显示了QWidget的基类名称为QWidget。

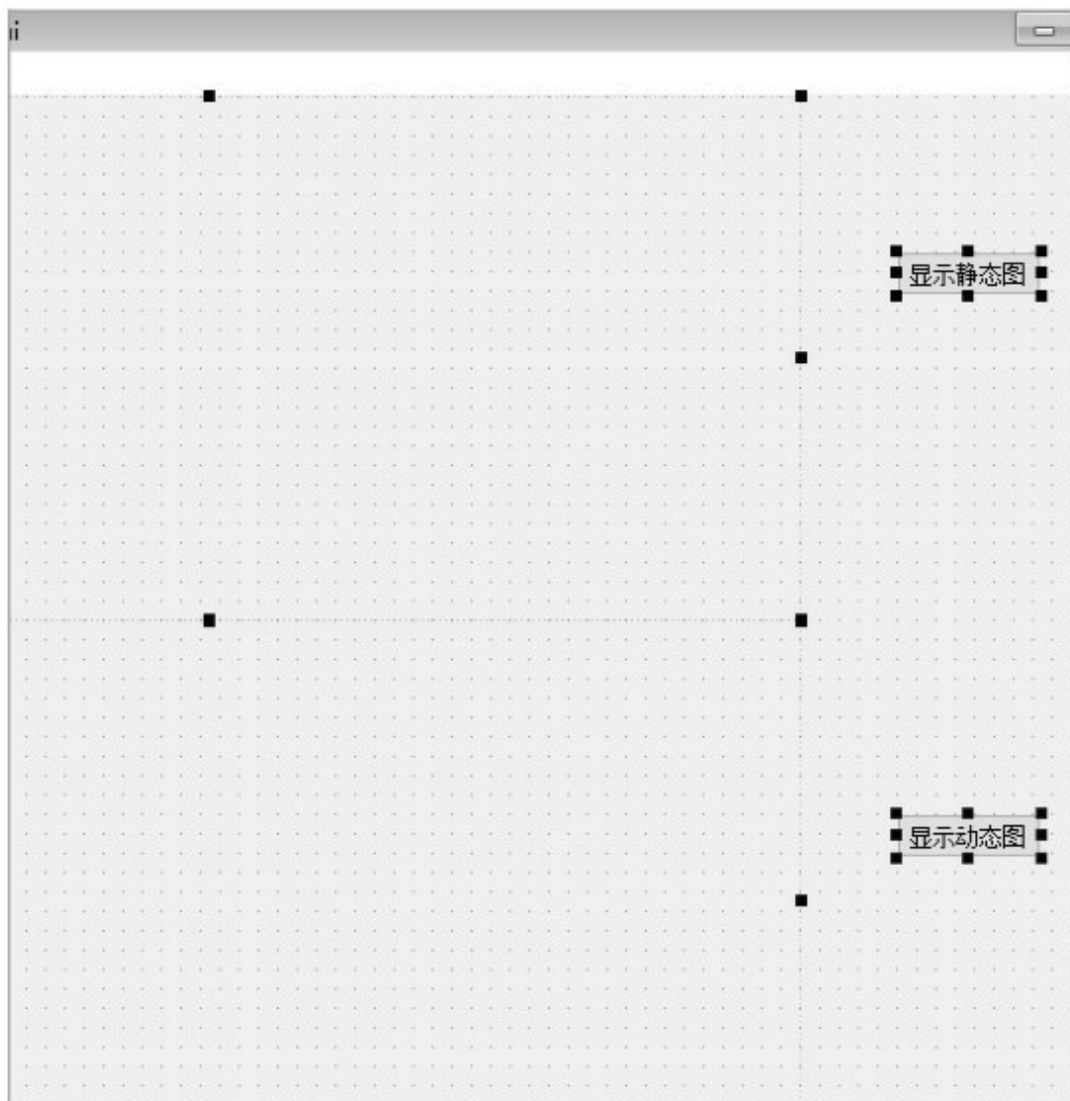


图9-31

```

    def button_click(self):
        # 显示静态图
        self.static_fig.show()
        # 显示动态图
        self.dynamic_fig.show()
    def __init__(self):
        self.setWindowTitle('MatplotlibWidget')
        self.layout = QVBoxLayout()
        self.layout.addWidget(self.static_fig)
        self.layout.addWidget(self.dynamic_fig)
        self.setLayout(self.layout)

```

### 9.4.3 MatplotlibWidget

```

#####
#####matplotlib_pyqt.py#####
class MainWindow(QMainWindow,Ui_MainWindow):
    def __init__(self,parent=None):
        super(MainWindow,self).__init__(parent)
        self.setupUi(self)
        self.matplotlibwidget_dynamic.setVisible(False)
        self.matplotlibwidget_static.setVisible(False)
#####——#####
    @pyqtSlot()
    def on_pushButton_clicked(self):
        """
        Slot documentation goes here.
        """
        self.matplotlibwidget_static.setVisible(True)
        self.matplotlibwidget_static.mpl.start_static_plot()
    @pyqtSlot()
    def on_pushButton_2_clicked(self):
        """
        Slot documentation goes here.
        """
        self.matplotlibwidget_dynamic.setVisible(True)
        self.matplotlibwidget_dynamic.mpl.start_dynamic_
plot()
#####
if __name__=="__main__":
    import sys

```



```

app=QApplication(sys.argv)
ui=MainWindow()
ui.show()
sys.exit(app.exec_())

```

图9-32 PyQt5 静态图与动态图

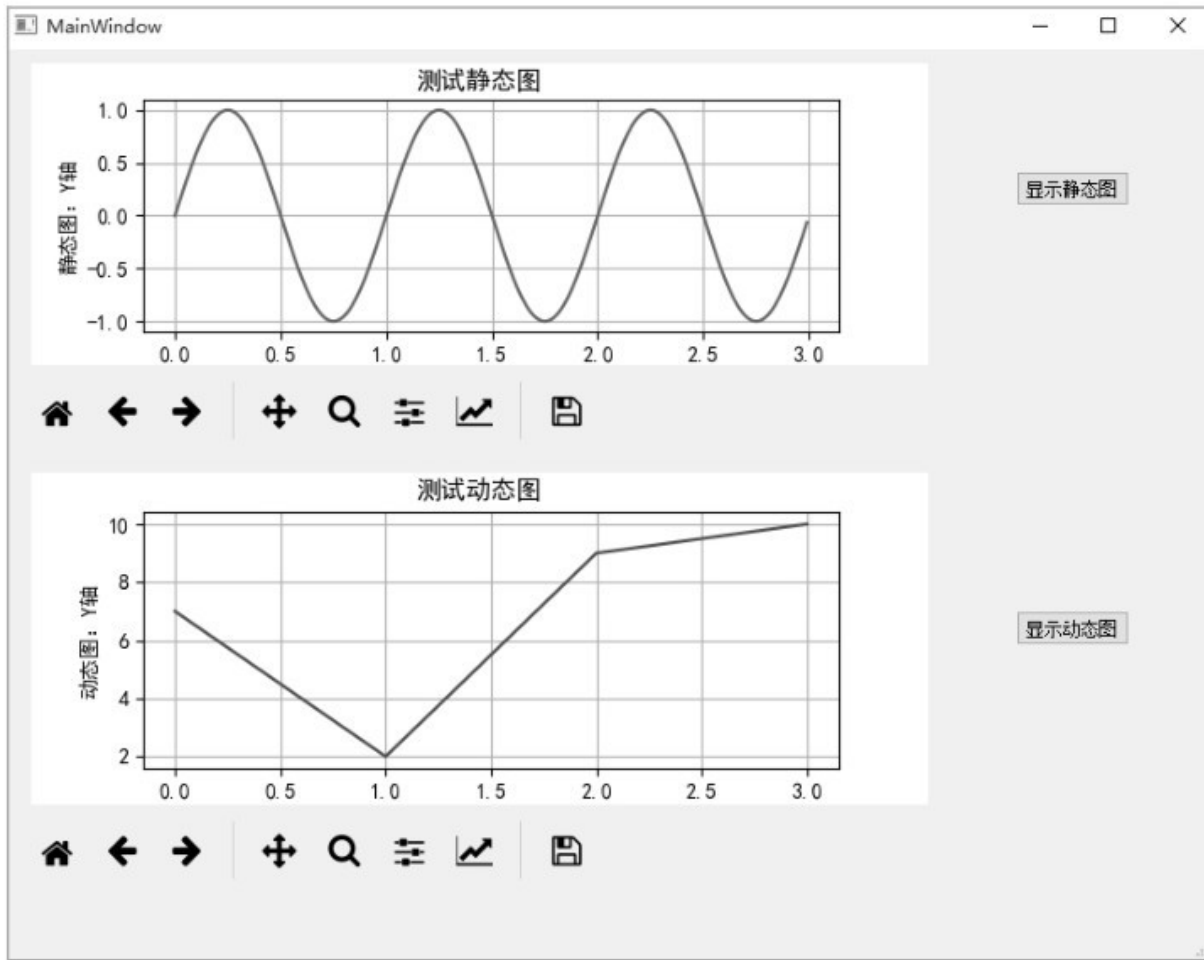
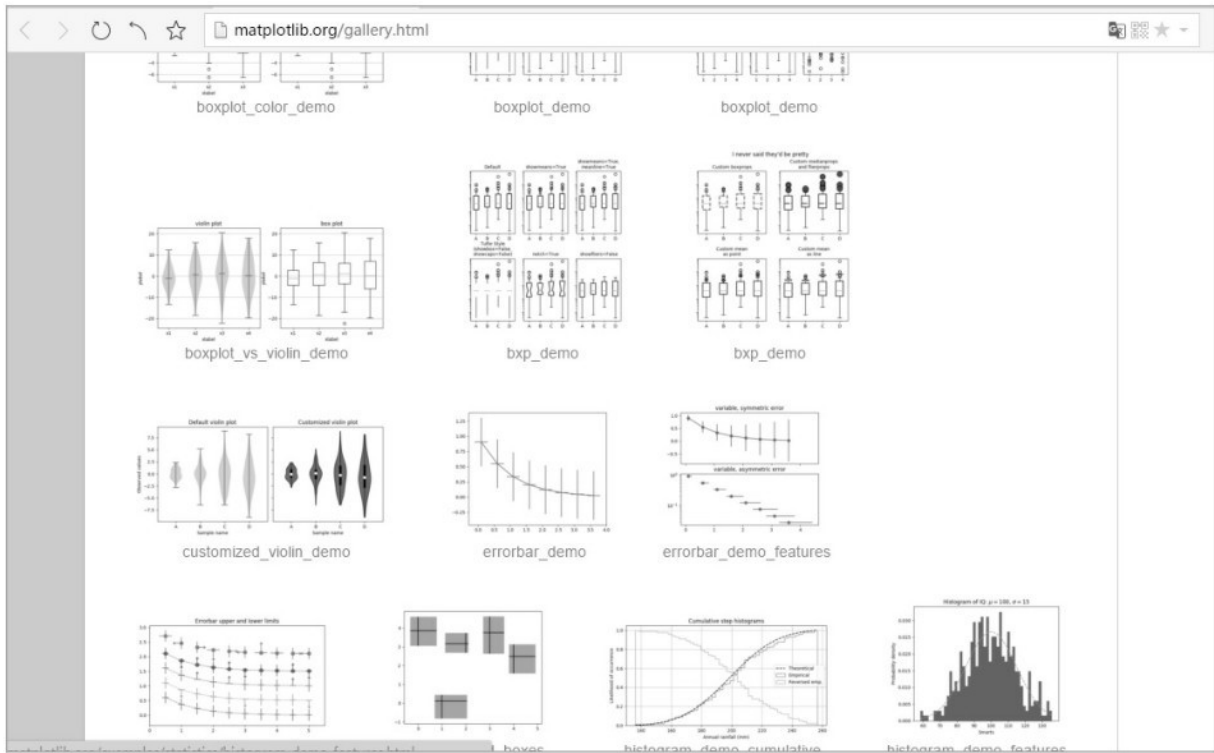


图9-32

本章主要介绍Matplotlib与PyQt5的结合使用，通过本章的学习，读者可以掌握Matplotlib与PyQt5的结合使用，为后续章节的学习打下基础。

## 9.4.4 本章小结

<http://matplotlib.org/gallery.html> Matplotlib  
 9-33 MyMplCanvas PyQt  
 Matplotlib



9-33

## [9.5 PyQtGraphPyQt](#)

PyQtGraphPythonGUIPyQtPySide  
 NumPy  
 PyQtGraphMITPyQtGraph

- 
-

PyQtGraphMatplotlibPyQtGraphMatplotlib  
MatplotlibPyQtGraph  
PyQtGraphPyQt  
PyQtGraphPyQt

### 9.5.1 PyQtGraph

PyQtGraphpip  
pip install pyqtgraph

### 9.5.2

PyQtGraph  
import pyqtgraph.examples  
pyqtgraph.examples.run()

9-34

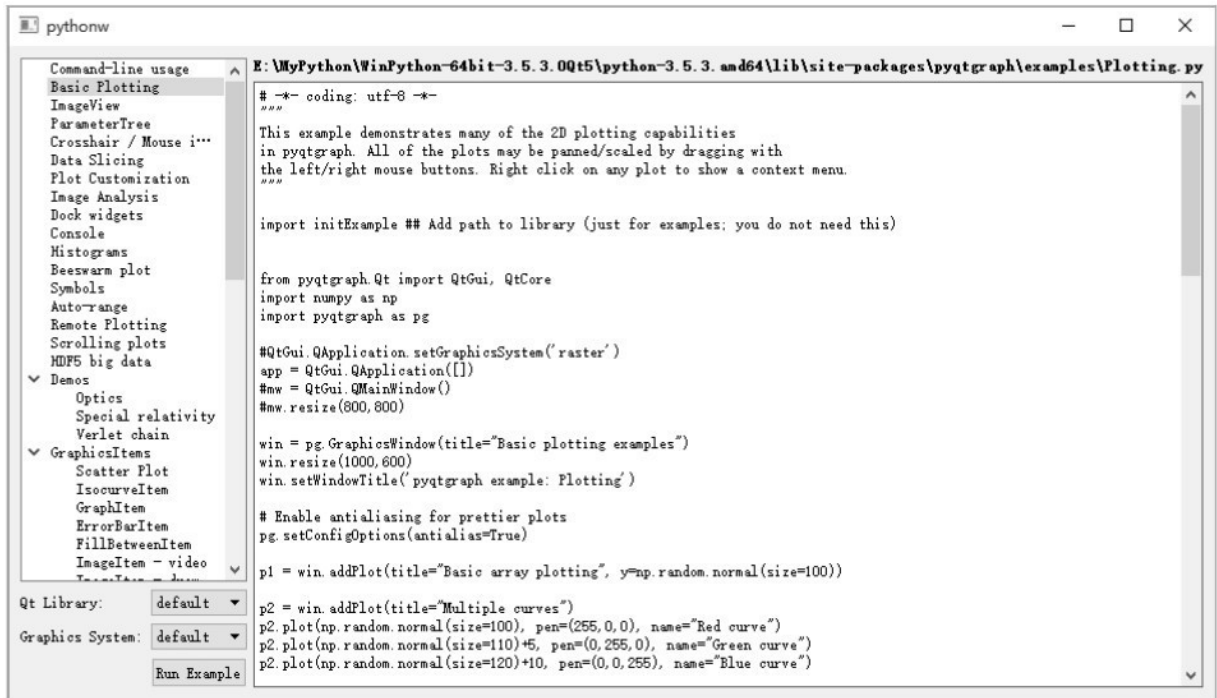
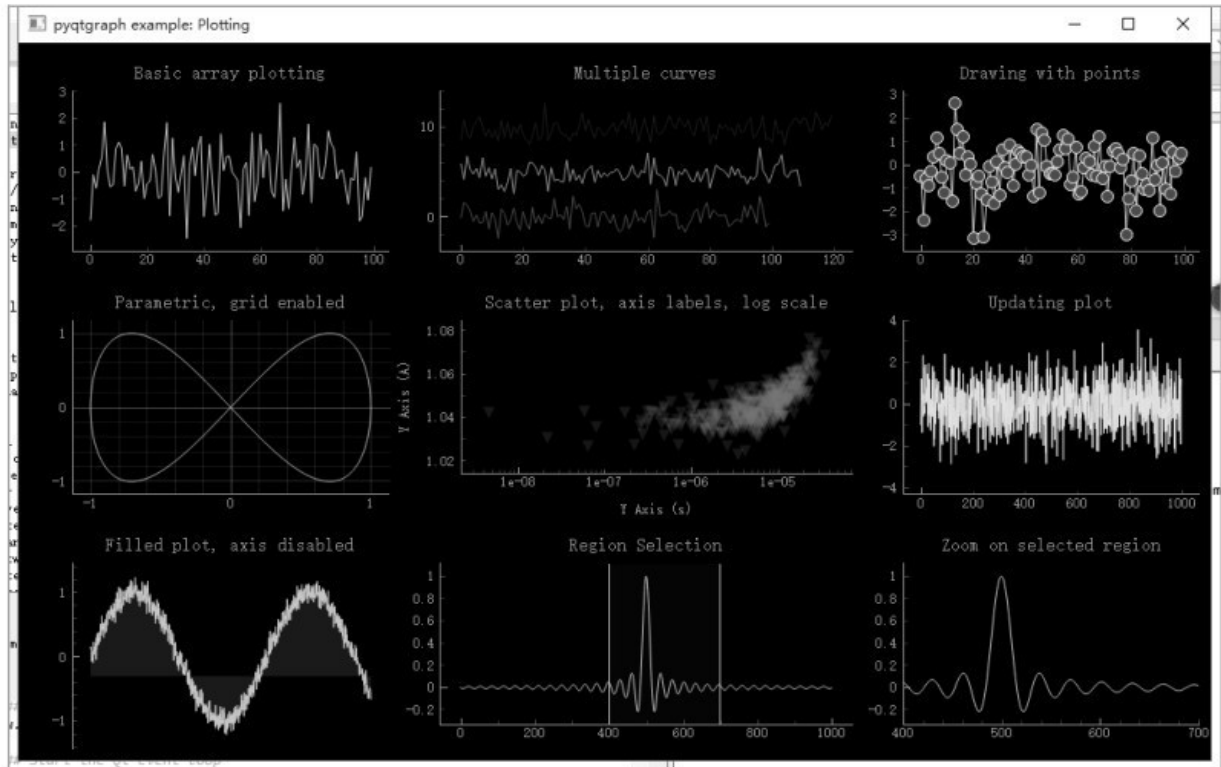


Figure 9-34

Figure 9-34 shows the "Basic Plotting" example. The "Run Example" button is highlighted. The example shows a window titled "Basic plotting examples" with three plots: "Basic array plotting", "Multiple curves", and "Scatter Plot". The "Multiple curves" plot shows three data series: "Red curve", "Green curve", and "Blue curve".



9-35

```

import pyqtgraph as pg
import numpy as np
win=pg.GraphicsWindow(title="Basic plotting
examples")
p1=win.addPlot(title="Basic array plotting",
y=np.random.normal(size=100))

```

```

import sys
from PyQt5.QtWidgets import QApplication, QMainWindow
from PyQt5.QtCore import Qt
from PyQt5.QtGui import QFont
from pyqtgraph.Qt import QtCore, QtGui
import pyqtgraph as pg
import numpy as np

```

```

p1=win.addPlot(title="Basic array plotting",
y=np.random.normal(size=100))

```

```

if __name__ == '__main__':
    app = QApplication(sys.argv)
    win.show()
    sys.exit(app.exec_())

```

### 9.5.3 PyQt5

```

from PyQt5.QtWidgets import QWidget, QVBoxLayout, QLabel, QLineEdit, QPushButton

```

9-36

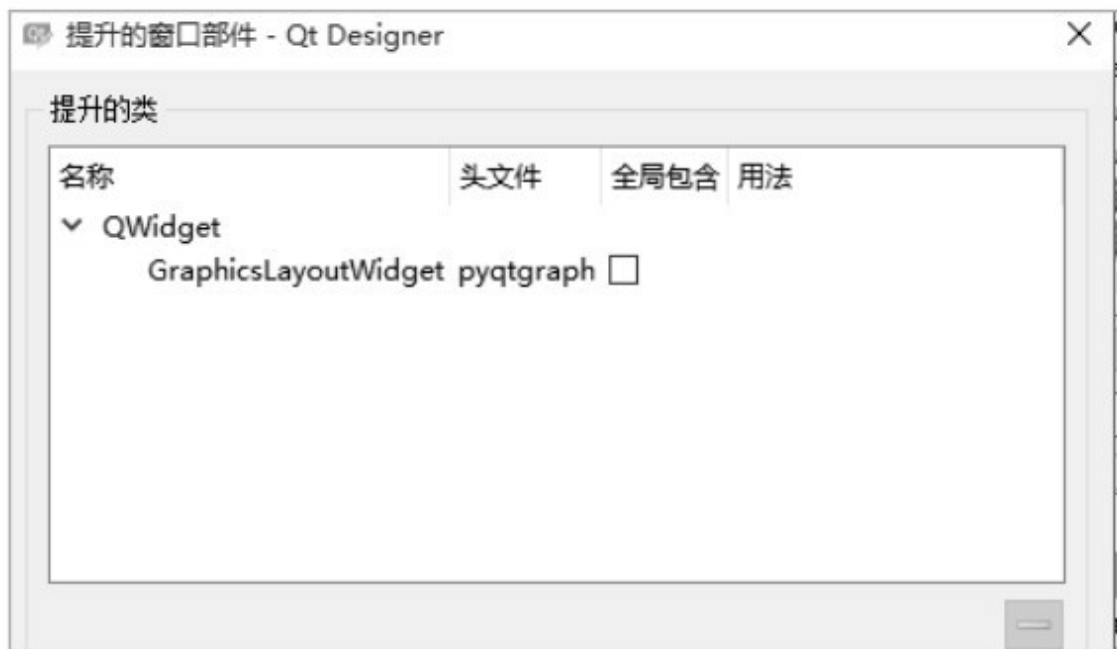


图9-36

在代码中，我们使用 `pyqtgraph1` 和 `pyqtgraph2` 来创建两个图形窗口。  
 图9-37展示了运行结果。



```
pg.setConfigOption('background','#f0f0f0') # 背景色  
pg.setConfigOption('foreground','d') # 前景色  
pg.setConfigOptions(antialias=True) # 抗锯齿  
# pg.setConfigOption('antialias',True) # 抗锯齿  
pg.setConfigOptions(font=font,font_size=font_size,  
setConfigOption(background_color='white')  
self.setupUi(self)  
  
pg.setStyleSheet("background-color: white; color: black; font-family: sans-serif; font-size: 16px; font-weight: bold; padding: 10px; border: 1px solid black;")  
pg.setWindowTitle("Pick Screen Color")  
pg.show() # 显示窗口
```





图9-38

□□□□□□□□□□□□

```
@pyqtSlot()
def on_pushButton_clicked(self):
    self.pyqtgraph1.clear() # 清空里面的内容，否则会发生重复绘图的结果

    '''第一种绘图方式'''
    self.pyqtgraph1.addPlot(title="绘制单条线",
y=np.random.normal(size=100), pen=pg.mkPen(color='b', width=2))

    '''第二种绘图方式'''
    plt2 = self.pyqtgraph1.addPlot(title='绘制多条线')

    plt2.plot(np.random.normal(size=150), pen=pg.mkPen(color='r',
```

```
width=2), name="Red curve") # pg.mkPen 的使用方法, 设置线条颜色为红色, 宽度为 2
    plt2.plot(np.random.normal(size=110) + 5, pen=(0, 255, 0),
name="Green curve")
    plt2.plot(np.random.normal(size=120) + 10, pen=(0, 0, 255),
name="Blue curve")
```

PyQtGraph 的 addPlot 方法

pg.mkPen 和 Qt.QPen 的区别  
<http://www.pyqtgraph.org/documentation/functions.html#pyqtgraph.mkColor>





## 9.6.1 Plotly

Plotly를 설치하기 위해 pip를

pip install plotly

## 9.6.2

Plotly는 GUI를 제공하며 Plotly를 사용하여  
Plotly를 PyQt5를 사용하여 PyQt5의 QWebView를 사용하여 Plotly PyQt5  
를

QWebView를 PyQt 5.7를 사용하여  
PyQt 5.6를 사용하여 QWebView를 QWebView를  
WebKit를 사용하여 JavaScript를 사용하여 Plotly를  
QWebView를 Chromium을 사용하여 Chrome  
QWebView를 PyQt를 사용하여

PyQt 5.7를 사용하여 9.6.6 “Plotly PyQt 5.6  
” PyQt 5.6를 사용하여 Plotly를

QWebView Plotly PyQt5/Chapter09/demo\_plotly\_pyqt.py

```
# -*- coding: utf-8 -*-  
  
"""  
Module implementing MainWindow.  
"""
```

```

from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
import sys
from PyQt5.QtWebEngineWidgets import QWebEngineView

class Window(QWidget):
    def __init__(self):
        QWidget.__init__(self)
        self.qwebengine = QWebEngineView(self)
        self.qwebengine.setGeometry(QRect(50, 20, 1200, 600))
        self.qwebengine.load(QUrl.fromLocalFile('\plotly_html\if_hs300_bais.html'))

app = QApplication(sys.argv)
screen = Window()
screen.showMaximized()
sys.exit(app.exec ())

```

□□□□□□□□

```

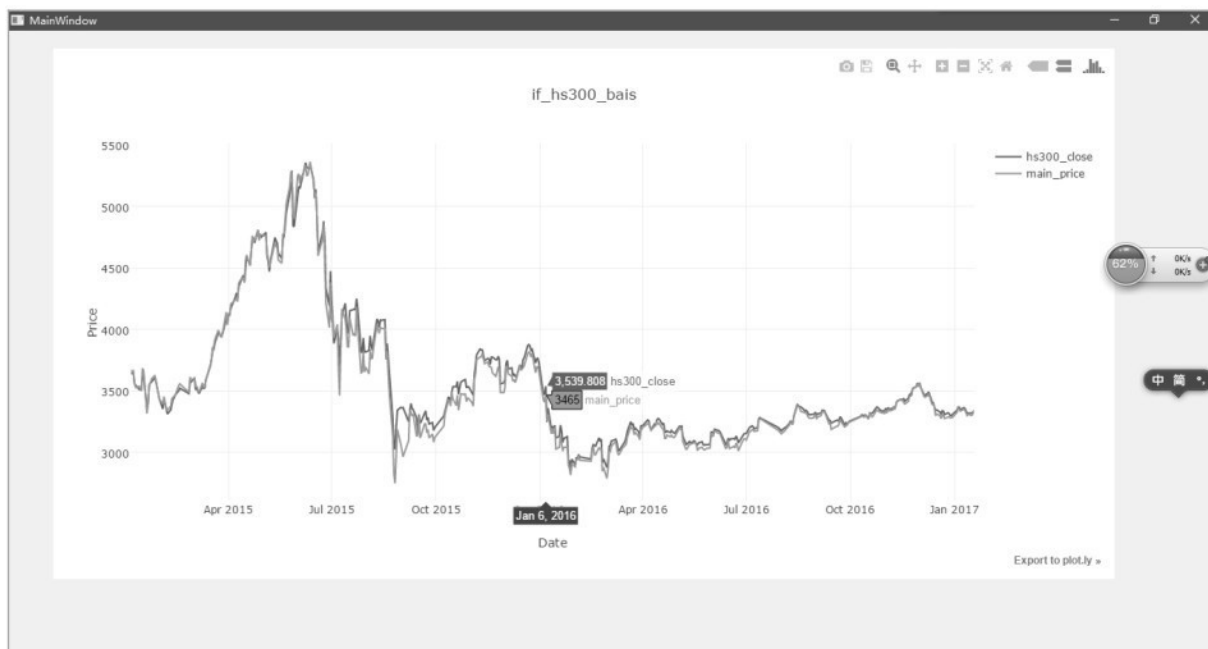
self.qwebengine=QWebEngineView(self)
self.qwebengine.load(QUrl.fromLocalFile('\plotly_html\if_
_hs300_bais.html'))

```

□□□□□□QWebEngineView□□□□QWebEngineView□□□□□□

□□□□if\_hs300\_bais.html□□Plotly□□□HTML□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□9-39□□□



9-39

Plotly 的“autoscale”选项

“autoscale”选项

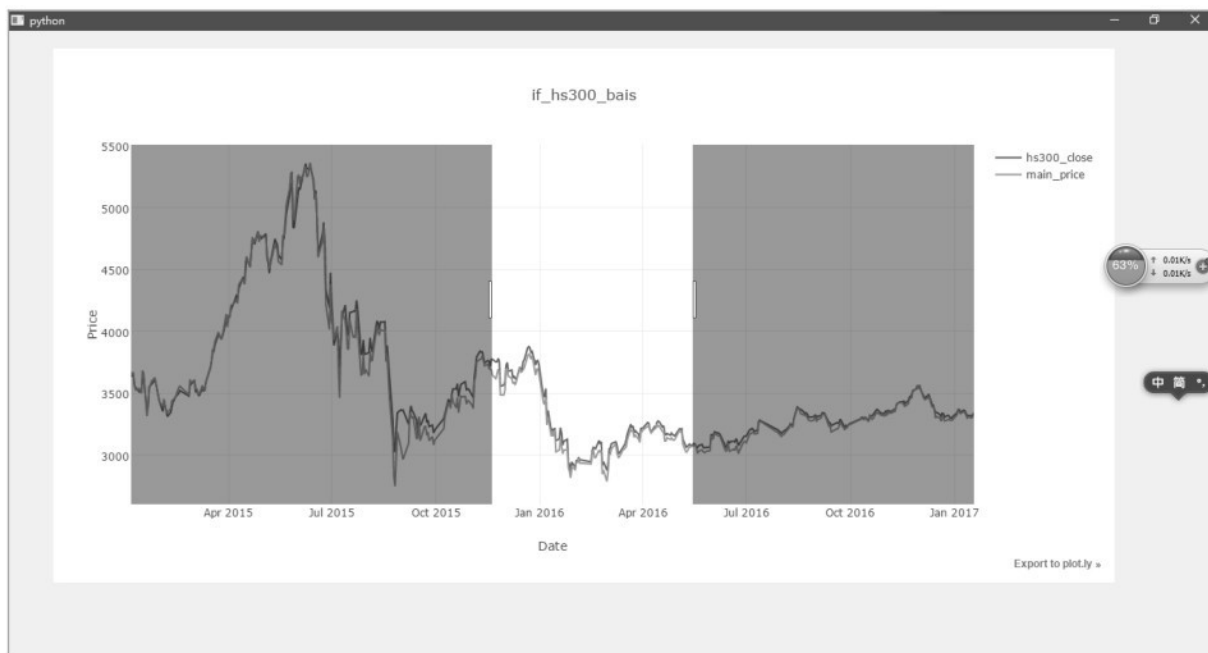


图9-40

### 9.6.3 数据可视化

在Qt Designer中使用QWebEngineView组件可以实现网页浏览功能。

在Qt Designer中使用QWidget组件可以实现自定义控件功能。

图9-41



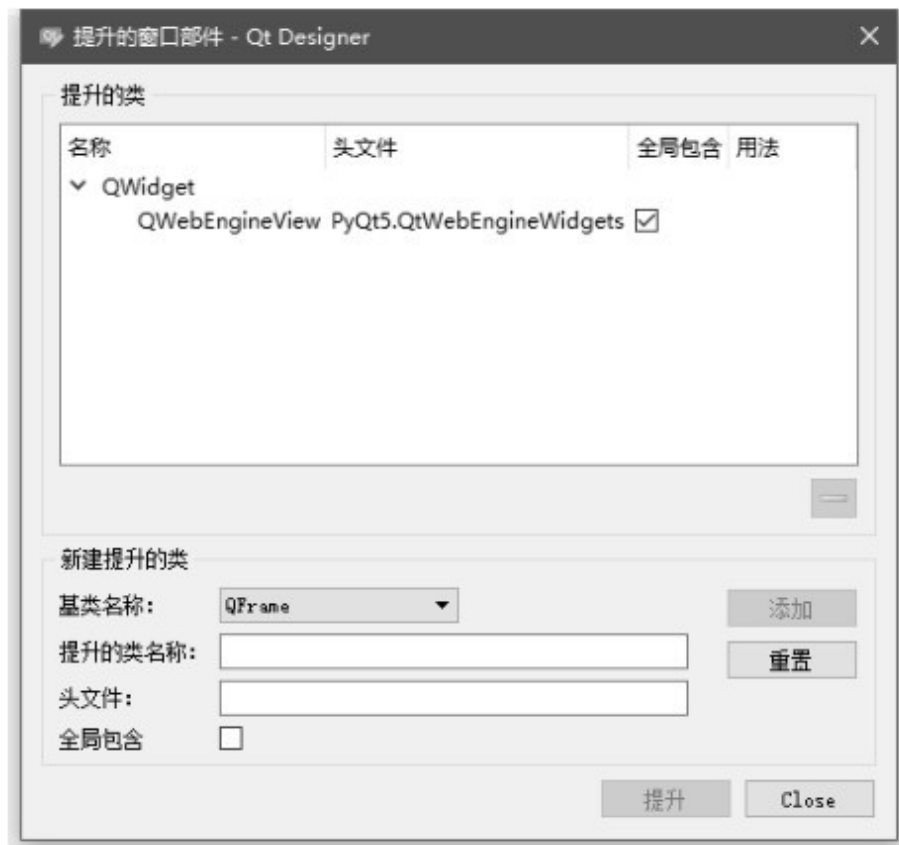


图9-41

## 9.6.4 Plotly\_PyQt5

在本节中，我们将使用QWebEngineView来显示Plotly生成的HTML内容。我们将使用PyQt 5的Plotly库来生成HTML内容。我们将使用Plotly的HTML输出功能来生成HTML内容。我们将使用Plotly\_PyQt5.py来生成HTML内容。

```

import pandas as pd
import os
import plotly.offline as pyof
import plotly.graph_objs as go

import numpy as np
import matplotlib.pyplot as plt

class Plotly_PyQt5():
    def __init__(self):
        '''初始化时设置存储 HTML 文件的文件夹名称，默认为 plotly_html'''
        plotly_dir = 'plotly_html'
        if not os.path.isdir(plotly_dir):
            os.mkdir(plotly_dir)

        self.path_dir_plotly_html = os.getcwd() + os.sep + plotly_dir

    def
get_plotly_path_if_hs300_bais(self, file_name='if_hs300_bais.html'):
    path_plotly = self.path_dir_plotly_html + os.sep + file_name
    df = pd.read_excel(r'if_index_bais.xlsx')

    '''绘制散点图'''
    line_main_price = go.Scatter(
        x=df.index,
        y=df['main_price'],
        name='main_price',
        connectgaps=True, # 这个参数表示允许连接数据之间的缺失值
    )

    line_hs300_close = go.Scatter(
        x=df.index,
        y=df['hs300_close'],
        name='hs300_close',
        connectgaps=True,

```

```

    )
    data = [line_hs300_close,line_main_price]

    layout = dict(title='if_hs300_bais',
                  xaxis=dict(title='Date'),
                  yaxis=dict(title='Price'),

    )
    fig = go.Figure(data=data, layout=layout)

    pyof.plot(fig, filename=path_plotly, auto_open=False)
    return path_plotly

```

```

def plotly_offline():
    """
    1. 创建Figure对象
    """
    import plotly.offline as pyof
    """
    2. 设置Figure对象的auto_open属性为False
    """
    pyof.plot(fig,filename=path_plotly,auto_open=False)
    """
    3. 返回Figure对象的path_plotly属性
    """
    return path_plotly
"""
PyQt5与plotly_pyqt.py
"""

```

```
# -*- coding: utf-8 -*-

"""
Module implementing MainWindow.
"""

from PyQt5.QtCore import pyqtSlot
from PyQt5.QtWidgets import QMainWindow

from Ui_plotly_pyqt import Ui_MainWindow


from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
import sys
```

```

from Plotly_PyQt5 import Plotly_PyQt5

class MainWindow(QMainWindow, Ui_MainWindow):
    """
    Class documentation goes here.
    """

    def __init__(self, parent=None):
        """
        Constructor

        @param parent reference to the parent widget
        @type QWidget
        """
        super(MainWindow, self).__init__(parent)
        self.setupUi(self)
        self.plotly_pyqt5 = Plotly_PyQt5()
        self.qwebengine.setGeometry(QRect(50, 20, 1200, 600))
        self.qwebengine.load(QUrl.fromLocalFile(
            self.plotly_pyqt5.get_plotly_path_if_hs300_bais()))

app = QApplication(sys.argv)
win = MainWindow()
win.showMaximized()
app.exec_()

```

□□□□□□□□

```

self.plotly_pyqt5=Plotly_PyQt5()
self.qwebengine.setGeometry(QRect(50,20,1200,600))
self.qwebengine.load(QUrl.fromLocalFile(self.plotly_pyq
t5.get_plotly_path_if_hs300_bais()))

```

□□□□□□□□□□□□□□□□□□

```

self.qwebengine.load(QUrl.fromLocalFile('\if_hs300_bais
.html'))

```

□□□□□□9-42□□□□□□9-39□□□□□□□□□□□□

Plotly  
QWebView  
QWebEngineView

MainWindow

if\_hs300\_bais

Price

5500

5000

4500

4000

3500

3000

Apr 2015 Jul 2015 Oct 2015 Jan 6, 2016 Apr 2016 Jul 2016 Oct 2016 Jan 2017

Date

hs300\_close

main\_price

3,539.808

3,468

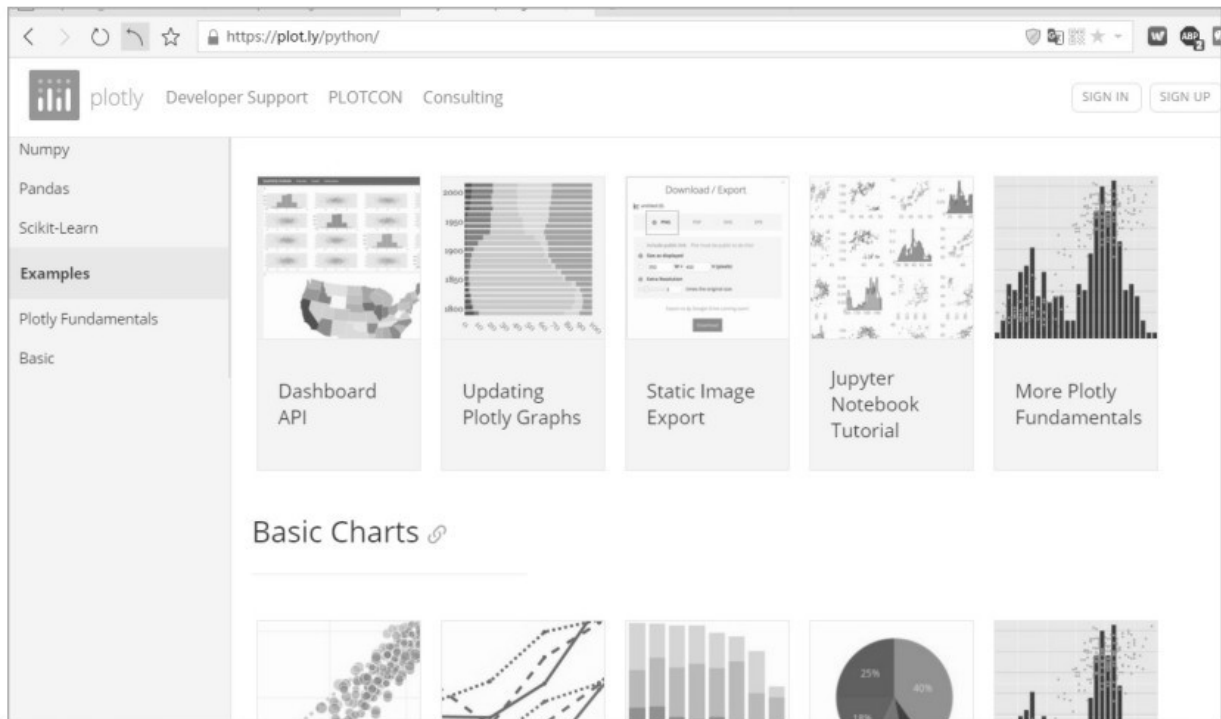
82%

中

Export to plot.ly »

### 9.6.5 □□□□

Plotly <https://plot.ly/python/>



9-43

get\_plotly\_path\_if\_hs300\_bais

## 9.6.6 Plotly PyQt 5.6

Plotly JavaScript PyQt 5.6  
 QWebView Plotly — Matplotlib  
 Plotly QWebView  
 PyQt 5.6 Plotly

PyQt Plotly  
 1 Matplotlib Matplotlib  
 Plotly Plotly Matplotlib





```

# -*- coding: utf-8 -*-

"""
Module implementing MainWindow.
"""

from PyQt5.QtCore import pyqtSlot
from PyQt5.QtWidgets import QMainWindow

from Ui_plotly_matplotlib_pyqt import Ui_MainWindow

from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
import sys
from Plotly_PyQt5 import Plotly_PyQt5

class MainWindow(QMainWindow, Ui_MainWindow):
    """
    Class documentation goes here.
    """

    def __init__(self, parent=None):
        """
        Constructor

        @param parent reference to the parent widget
        @type QWidget
        """
        super(MainWindow, self).__init__(parent)
        self.setupUi(self)
        self.plotly_pyqt5 = Plotly_PyQt5()
        self.webView.setGeometry(QRect(50, 20, 1200, 600))
        self.webView.load(QUrl.fromLocalFile
        (self.plotly_pyqt5.get_plot_path_matplotlib_plotly()))

app = QApplication(sys.argv)
win = MainWindow()

```

```
win.showMaximized()
app.exec ()
```

□□□□□□□□

```
self.plotly_pyqt5=Plotly_PyQt5()
self.webView.setGeometry(QRect(50,20,1200,600))
self.webView.load(QUrl.fromLocalFile(self.plotly_pyqt5.get_plot_path_matplotlib_plotly()))
```

□□□□QWebView □□□□□□□□□□ QWebEngineView □□□□□□□□  
 □□□□□□□□□□Plotly\_PyQt5□□get\_plot\_path\_matplotlib\_plotly□□  
 □□□□□□□□□□□□□□□□□□

```
def get_plot_path_matplotlib_plotly(self,
    file_name='matplotlib_plotly.html'):
    path_plotly = self.path_dir_plotly_html + os.sep + file_name

    N = 50
    x = np.random.rand(N)
    y = np.random.rand(N)
    colors = np.random.rand(N)
    area = np.pi * (15 * np.random.rand(N)) ** 2 # 0 to 15 point radii
    scatter_mpl_fig = plt.figure()
    plt.scatter(x, y, s=area, c=colors, alpha=0.5)

    pyof.plot_mpl(scatter_mpl_fig, filename=path_plotly,
        resize=True, auto_open=False)
    return path_plotly
```

□□□□□□□□

□1□□□□□□□□□□plot\_mpl□□□□□□□□plot□□□plot\_mpl□□□□□  
 Matplotlib□□□□□□□□□□Plotly□□□□□□□□□□  
 □2□□□□□□□□□□□□Matplotlib□□□□□□□□Plotly□□□□□□□□







1. HP 4 IATA  
 2. IATA  
 3. HP UFT (Unified Functional Testing)  
 4. IATA

Qt Designer Python  
 unittest PyQt 5 QTest UI

## [9.7.2](#)



9-47

9-47  
 Margarita Midori  
 8 Ei Charro Anejo 20 Midori 10 20  
 Triple Sec 20  
 Midori Triple Sec  
 jigger  
 0.0444

[illegible][illegible]

□9-48



□9-49

PyQt Qtest

- MargaritaMixer.ui Qt Designer XML

- MargaritaMixer.py GUI Python

pyuic5-o MatrixWinUi.py MatrixWinUi.ui

pyuic5-o MatrixWinUi.py MatrixWinUi.ui

- CallMatrixWinUi.py GUI

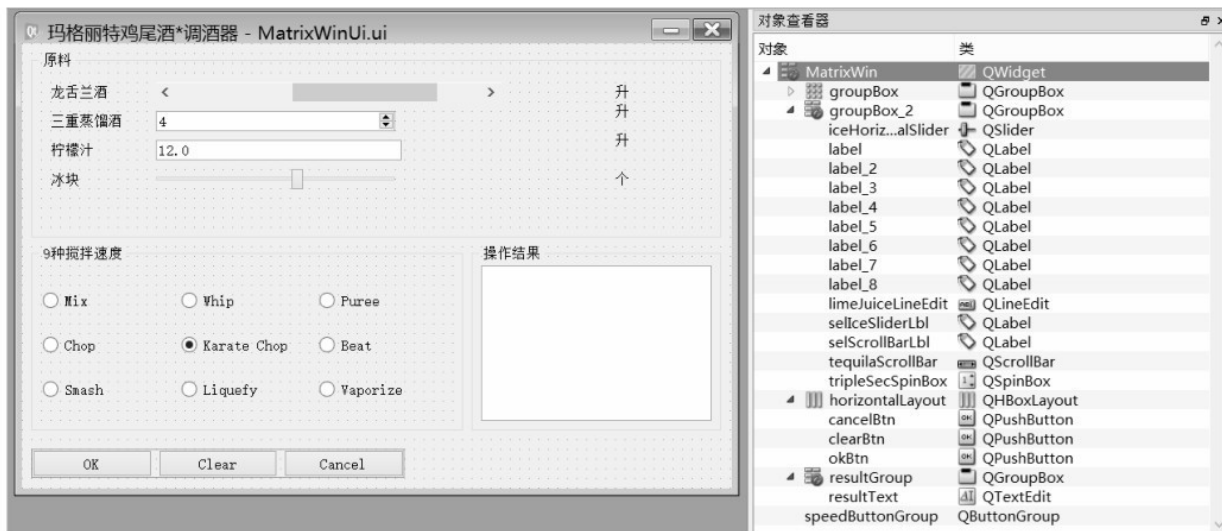
- MatrixWinTest.py

- RunTestCase.py

PyQt5/Chapter03/testCase

Qt Designer 9-50 9-51

MatrixWinUi.ui PyQt5/Chapter03/testCase





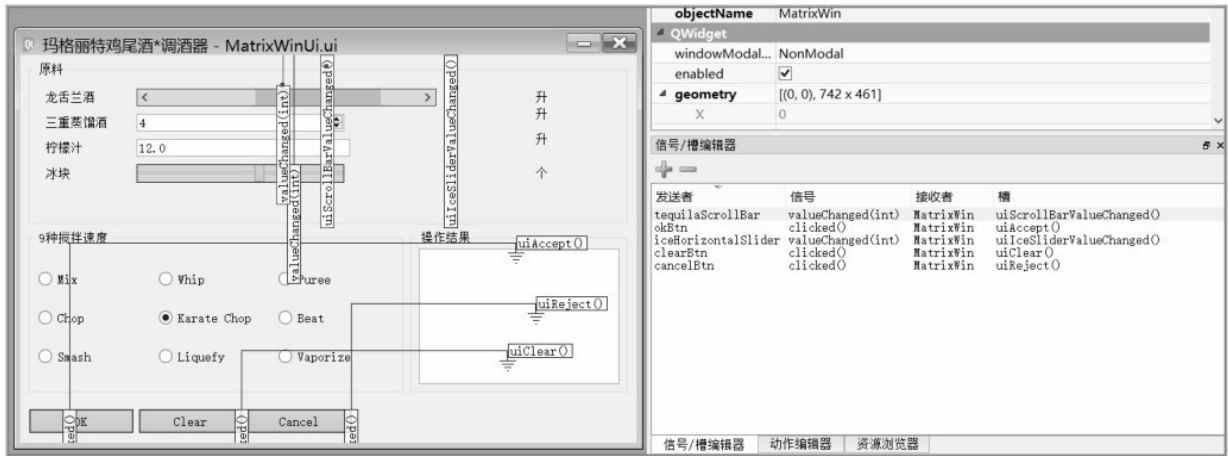


图9-51

0.0444“9”  
 “Ok”  
 “Clear”“”“Cancel”

9-6

图9-6



pyuic5-o MatrixWinUi.py MatrixWinUi.ui

□ □ □ □ □ □ □ □ □ □ □ □ □ □

PyQt5/Chapter03/testCase/MatrixWinUi.py□□□

CallMatrixWinUi.py

Qt DesignerMatrixWinUi.uipyuic5

CallMatrixWinUi.py

PyQt5/Chapter03/mainWin□□□□□□□□□□

```
import sys

from PyQt5.QtWidgets import *
from MatrixWinUi import *

class CallMatrixWinUi(QWidget ):

    def __init__(self, parent=None):
        super(CallMatrixWinUi, self).__init__(parent)
        self.ui = Ui_MatrixWin()
        self.ui.setupUi(self)
        self.initUi()

# 初始化窗口
def initUi(self):
    scrollVal = self.ui.tequilaScrollBar.value()
    self.ui.selScrollBarLbl.setText( str(scrollVal) )
    sliderVal = self.ui.iceHorizontalSlider.value()
```

```

        self.ui.selIceSliderLbl.setText( str(sliderVal) )

# 获得一个量酒器的重量, 单位: 克
def getJiggers(self):
    # 返回玛格丽特鸡尾酒的总容量, 以 jigger 量酒器为单位
    # 一个量酒器可以容纳 0.0444 升的酒
    jiggersTequila = self.ui.tequilaScrollBar.value()
    jiggersTripleSec = self.ui.tripleSecSpinBox.value()
    jiggersLimeJuice = float(self.ui.limeJuiceLineEdit.text())
    jiggersIce = self.ui.iceHorizontalSlider.value()
    return jiggersTequila + jiggersTripleSec + jiggersLimeJuice +
jiggersIce

# 获得一个量酒器的体积, 单位: 升
def getLiters(self):
    '''返回鸡尾酒的总容量(升)'''
    return 0.0444 * self.getJiggers()

# 获得搅拌速度
def getSpeedName(self):
    speedButton = self.ui.speedButtonGroup.checkedButton()
    if speedButton is None:
        return None
    return speedButton.text()

# 单击“OK”按钮后, 把响应的结果显示在 resultText 文本框里
def uiAccept(self):
    print('* CallMatrixWinUi accept ')
    print('The volume of drinks is {0} liters ({1}
jiggers)'.format(self.getLiters() , self.getJiggers() ))
    print('The blender is running at speed
"{0}"'.format(self.getSpeedName() ))
    msg1 = '饮料量为: {0} 升 ({1} 个量酒器)。
'.format(self.getLiters() , self.getJiggers() )
    msg2 = '调酒器的搅拌速度是: "{0}"'.format(self.getSpeedName() )
    self.ui.resultText.clear()
    self.ui.resultText.append(msg1)
    self.ui.resultText.append(msg2)

# 单击“Cancel”按钮, 关闭窗口
def uiReject(self):
    print('* CallMatrixWinUi reject ')

```

```

        '''Cancel.'''
        self.close()

# 单击“Clear”按钮，清空操作结果
def uiClear(self):
    print('* CallMatrixWinUi uiClear ')
    self.ui.resultText.clear()

def uiScrollBarValueChanged(self):
    print('* uiScrollBarValueChanged -----')
    pos = self.ui.tequilaScrollBar.value()
    self.ui.selScrollBarLbl.setText( str(pos) )

def uiIceSliderValueChanged( self):
    print('* uiIceSliderValueChanged -----')
    pos = self.ui.iceHorizontalSlider.value()
    self.ui.selIceSliderLbl.setText( str(pos) )

if __name__=="__main__":
    app = QApplication(sys.argv)
    demo = CallMatrixWinUi()
    demo.show()
    sys.exit(app.exec_())

```

□□□□□□□□□□9-52□□□

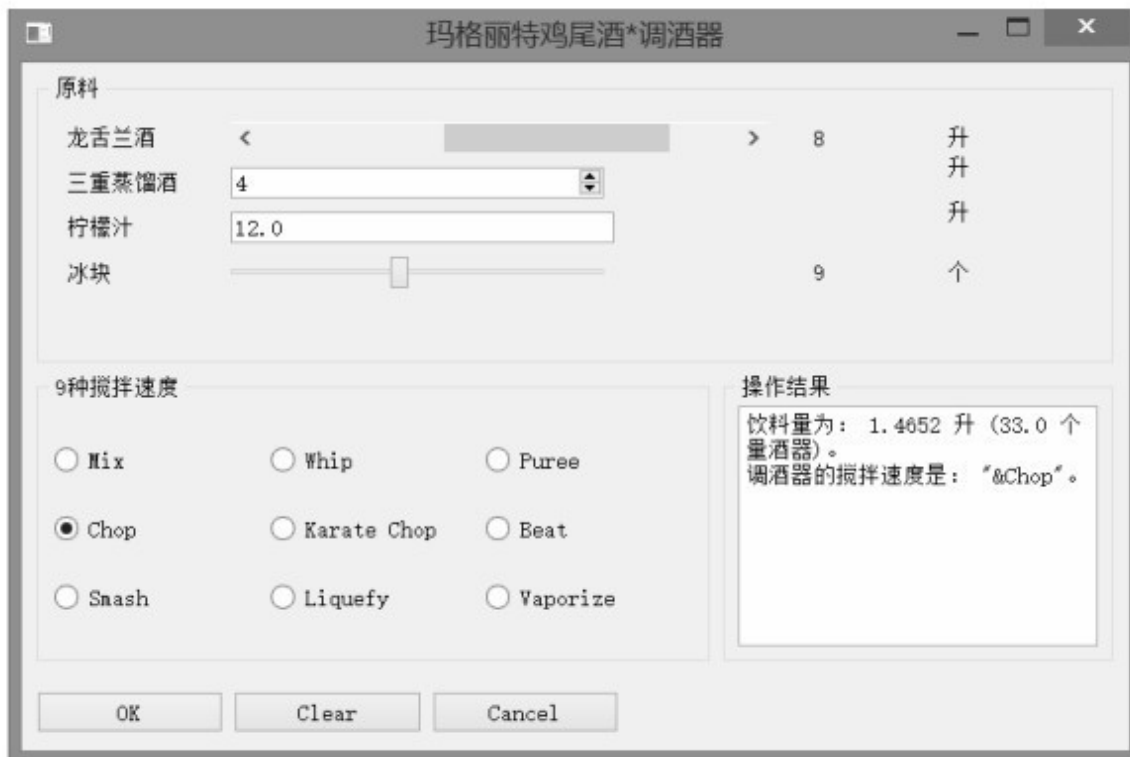


图9-52

### 9.7.3 测试案例

测试案例CallMatrixWinUi的测试案例文件MatrixWinTest.py位于PyQt5/Chapter03/testCase/CallMatrixWinUi.py文件中。

#### 1. 测试案例

测试案例MatrixWinTest使用Python的unittest模块。

1. 测试案例unittest

import unittest

2. 测试案例unittest.TestCase类继承MatrixWinTest

```

    3 setUp()tearDown()
setUp()tearDown()
    4 test test_moveScrollBar()
test_tripleSecSpinBox()
    5 assertEquals
assertRaises
    6 unittest.main()
    9-8

```

9-8

断言方法	说 明
assertEqual(a, b)	检测 a==b
assertNotEqual(a,b )	检测 a!=b
assertTrue(x)	检测 bool(x) is True
assertFalse(x )	检测 bool(y) is False
assertIsNot(a, b)	检测 a is not b

```

class MatrixWinTest(unittest.TestCase)
unittest

```

```

class MatrixWinTest(unittest.TestCase):
    #
    def setUp(self):
        print('*** setUp ***')
        self.app=QApplication(sys.argv)
        self.form=CallMatrixWinUi.CallMatrixWinUi()
        self.form.show()
    #
    def tearDown(self):
        print('*** tearDown ***')

```

```
self.app.exec_()
```

## 2. □□□□□□

PyQt 应用程序的入口点通常是 `main()` 函数。在这个函数中，我们通常会调用 `exit()` 或 `sys.exit()` 来结束程序。这确保了程序在运行结束后能够正确地退出。

```

    BackWorkThread
    PyQt
    QThread

```

```
# 1000 QThread 1
```

```
class BackWorkThread(QThread):
```

```
# [ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]str
```

```
finishSignal=pyqtSignal(str)
```

# □□□□□□□□

```
def __init__(self,sleepTime,parent=None):
```

```
super(BackWorkThread,self).__init__(parent)
```

# 

--	--	--	--

```
self.sleepTime=sleepTime
```

```
# run()
```

```
def run(self):
```

# 

--	--	--	--	--	--

```
time.sleep(self.sleepTime)
```

#

```
self.finishSignal.emit('ok ,begin to close Window')
```

```
MatrixWinTest.setUp()
BackWorkThread(5)
MatrixWinTest
```



```

class MatrixWinTest(unittest.TestCase):
    # 初始化工作
    def setUp(self):
        print('*** setUp ***')
        self.app = QApplication(sys.argv)
        self.form = CallMatrixWinUi.CallMatrixWinUi()

        self.form.show()

        # 新建线程对象，传入参数，每 5 秒关闭一个测试用例
        self.bkThread = BackWorkThread(int( 5 ))
        # 连接子进程的信号和槽函数
        self.bkThread.finishSignal.connect(self.closeWindow)
        #self.bkThread.finishSignal.connect(self.app.exec_)

        # 启动线程，开始执行 run() 函数中的内容
        self.bkThread.start()

    # 退出清理工作
    def tearDown(self):
        print('*** tearDown ***')
        self.app.exec_()

```

### 3. 测试用例

测试用例“01”测试矩阵窗口是否能正常关闭“9 测试用例”测试矩阵窗口  
 测试用例“02”测试矩阵窗口是否能正常关闭“OK”测试用例“03”测试

```

# 测试用例——在默认状态下测试 GUI
def test_defaults(self):
    '''测试 GUI 处于默认状态'''

    print('*** testCase test_defaults begin ***')
    self.form.setWindowTitle('开始测试用例 test_defaults ')

    self.assertEqual(self.form.ui.tequilaScrollBar.value(), 8)
    self.assertEqual(self.form.ui.tripleSecSpinBox.value(), 4)
    self.assertEqual(self.form.ui.limeJuiceLineEdit.text(), "12.0")
    self.assertEqual(self.form.ui.iceHorizontalSlider.value(), 12)
    self.assertEqual(self.form.ui.speedButtonGroup.
checkedButton().text(), "&Karate Chop")
    print('*** speedName='+ self.form.getSpeedName() )

    # 用鼠标左键单击“OK”按钮
    okWidget = self.form.ui.okBtn
    QTest.mouseClick(okWidget, Qt.LeftButton)

    #测试窗口在默认状态下，各控件的默认值是否与预期值一样
    self.assertEqual(self.form.getJiggers() , 36.0)
    self.assertEqual(self.form.getSpeedName(), "&Karate Chop")

    print('*** testCase test_defaults end ***')

```

□□

□□□□□□QTest.mouseClick()□□□□□□“OK”□□□

□□□□□□□□□□□□□□9-53□□□

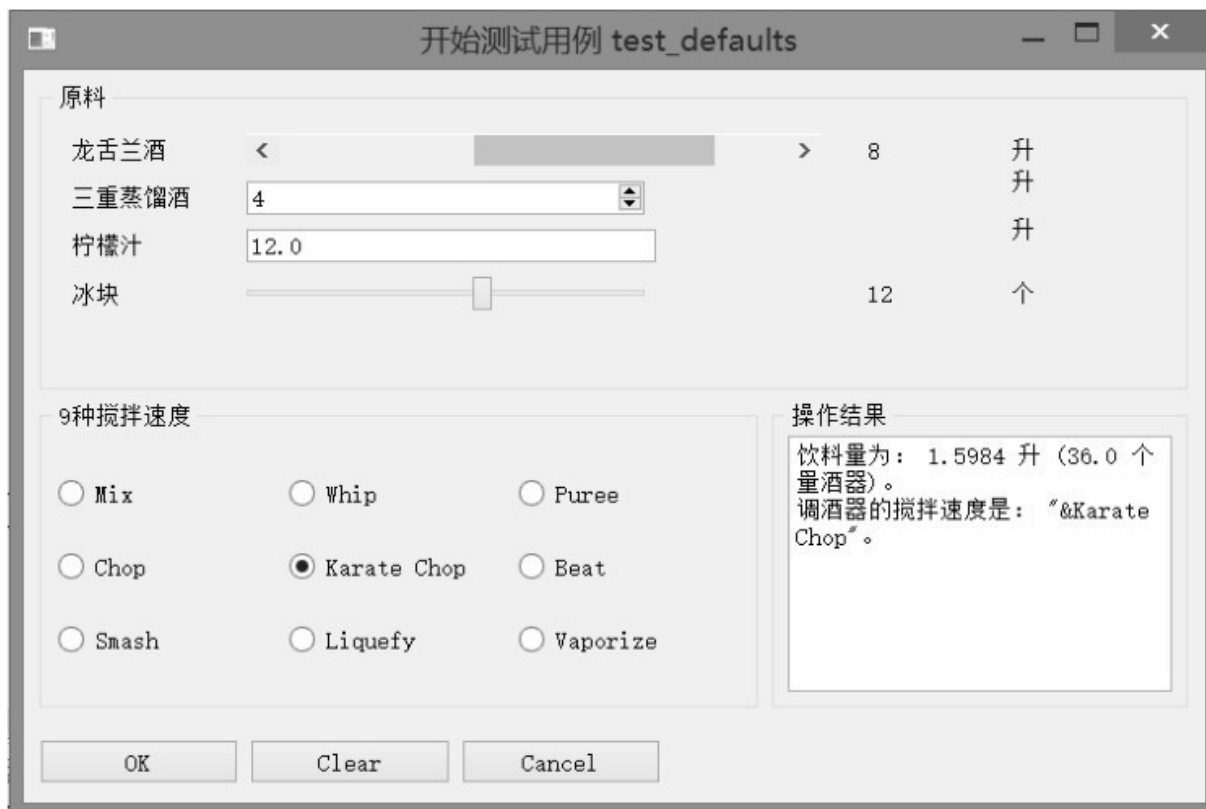
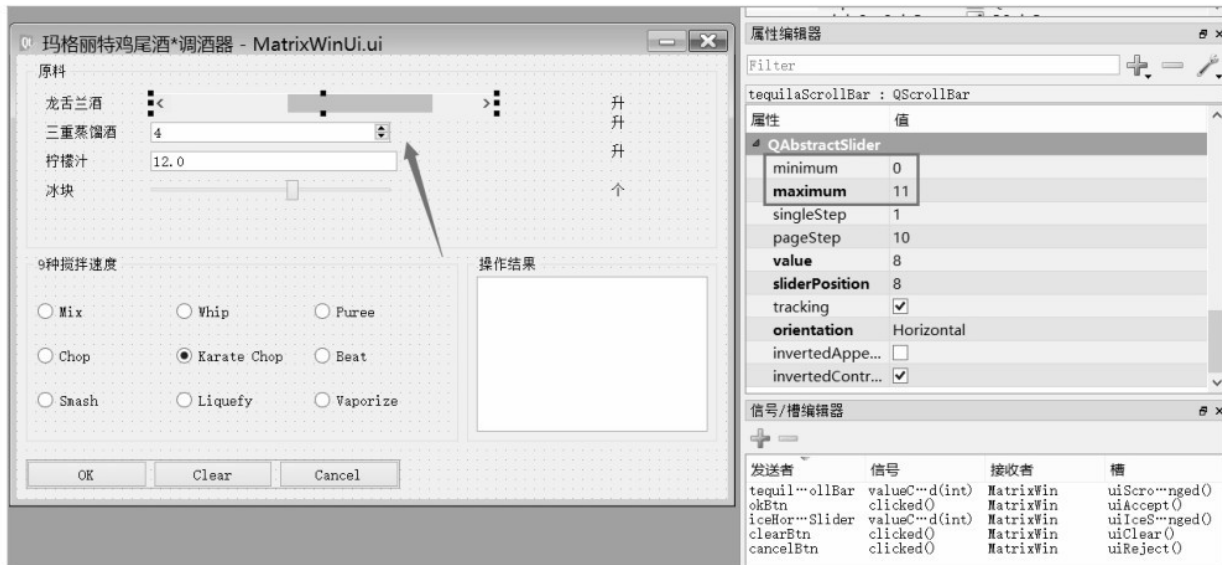


图9-53

## 4. PyQt的QScrollBar

QScrollBar是Qt提供的一个滚动条控件，它用于在窗口中显示滚动条。QScrollBar的构造函数如下所示，其中0表示滚动条的初始位置，12表示滚动条的最大值，1表示滚动条的最小值，0表示滚动条的初始位置，9-54表示滚动条的宽度。



9-54

def setFormToZero() # 重置所有控件为0

# 重置所有控件为0

def setFormToZero(self):

print('\* setFormToZero \*')

self.form.ui.tequilaScrollBar.setValue(0)

self.form.ui.tripleSecSpinBox.setValue(0)

self.form.ui.limeJuiceLineEdit.setText("0.0")

self.form.ui.iceHorizontalSlider.setValue(0)

self.form.ui.selScrollBarLbl.setText("0")

self.form.ui.sellIceSliderLbl.setText("0")

def setFormToZero(self):

print('\* setFormToZero \*')

```

# 测试用例——测试滑动条
def test_moveScrollBar(self):
    '''测试用例 test_moveScrollBar'''
    print('*** testCase test_moveScrollBar begin ***')
    self.form.setWindowTitle('开始测试用例 test_moveScrollBar ')
    self.setFormToZero()

    # 测试将龙舌兰酒的滑动条的值设置为 12，在 UI 中实际它的最大值为 11
    self.form.ui.tequilaScrollBar.setValue( 12 )
    print('* 当执行 self.form.ui.tequilaScrollBar.setValue(12) 后，
ui.tequilaScrollBar.value() => ' +
str( self.form.ui.tequilaScrollBar.value() ) )
    self.assertEqual(self.form.ui.tequilaScrollBar.value(), 11 )

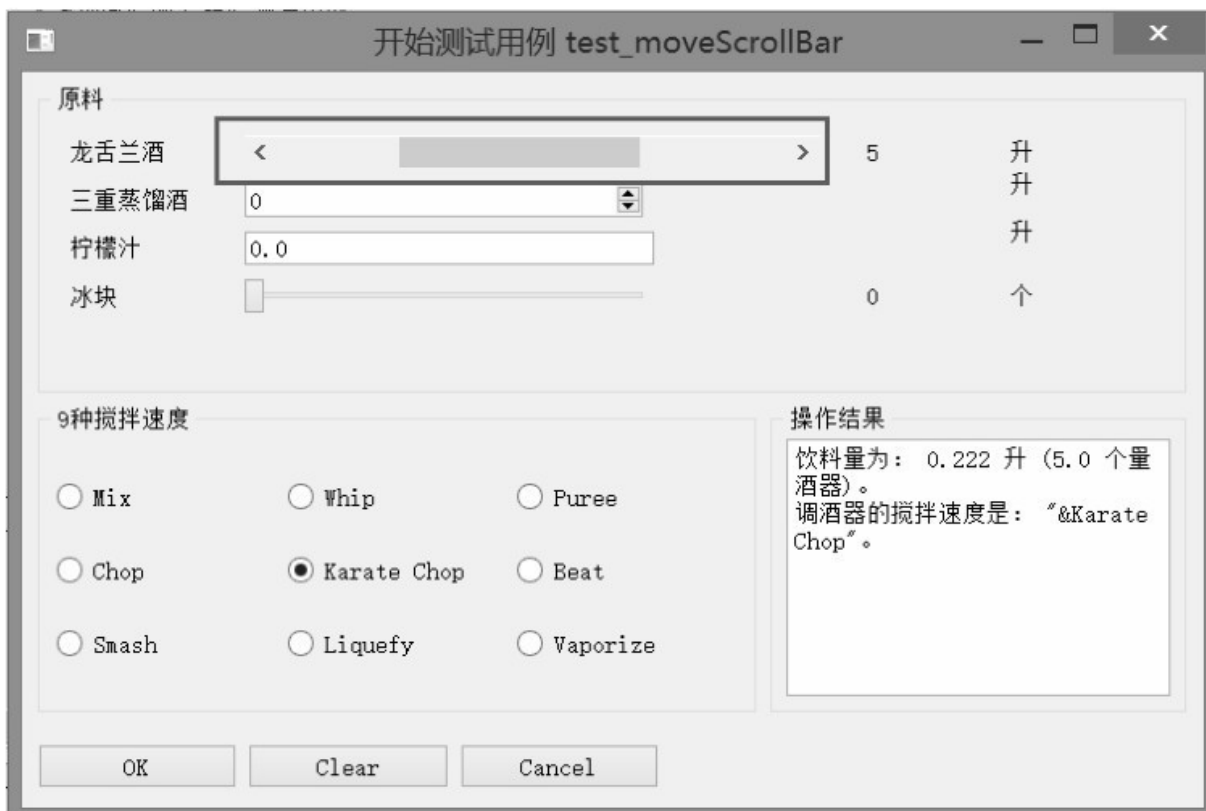
    # 测试将龙舌兰酒的滑动条的值设置为 -1，在 UI 中实际它的最小值为 0
    self.form.ui.tequilaScrollBar.setValue(-1)
    print('* 当执行 self.form.ui.tequilaScrollBar.setValue(-1) 后，
ui.tequilaScrollBar.value() => ' +
str( self.form.ui.tequilaScrollBar.value() ) )
    self.assertEqual(self.form.ui.tequilaScrollBar.value(), 0)

    # 重新将龙舌兰酒的滑动条的值设定置 5
    self.form.ui.tequilaScrollBar.setValue(5)

    # 用鼠标左键单击“OK”按钮
    okWidget = self.form.ui.okBtn
    QTest.mouseClick(okWidget, Qt.LeftButton)
    self.assertEqual(self.form.getJiggers() , 5)
    print('*** testCase test_moveScrollBar end ***')

```

□□□□□□□□□□□□□□9-55□□□□



□9-55

## 5. PyQt QSpinBox

QSpinBox 12-1 UI  
11 0  
9-56

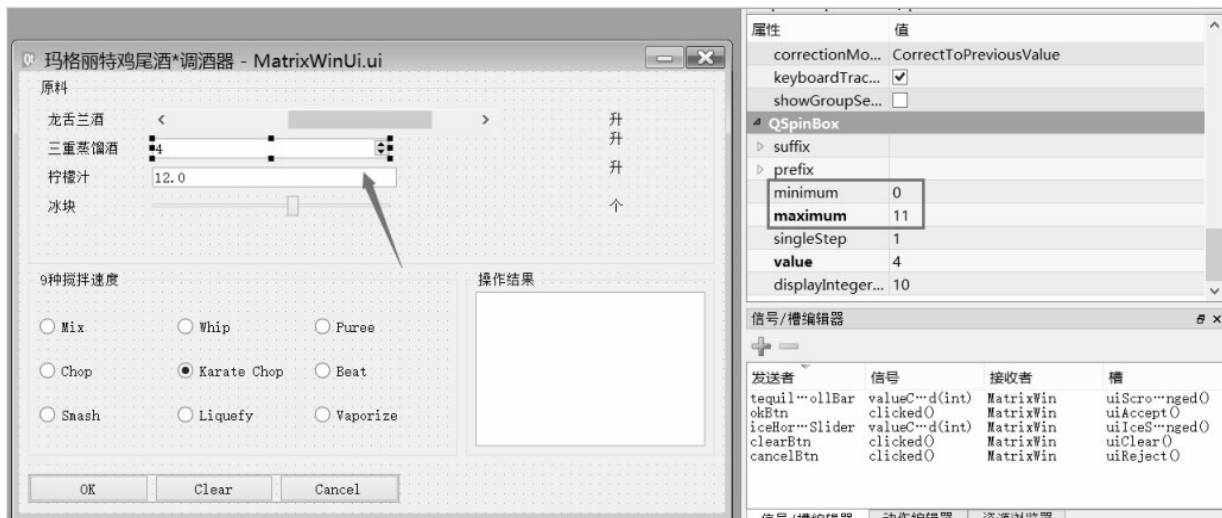


图9-56

# 测试用例——测试计数器控件 (QSpinBox)

```
def test_tripleSecSpinBox(self):
    '''测试用例 test_tripleSecSpinBox '''
    print('*** testCase test_tripleSecSpinBox begin ***')
    self.form.setWindowTitle('开始测试用例 test_tripleSecSpinBox ')
    '''测试修改计数器控件 (QSpinBox) 的最大值、最小值
        测试它的最小值和最大值作为读者的练习'''
```

```

'''
    self.setFormToZero()
    # tripleSecSpinBox 在界面中的取值范围为 0~11，将它的最大值设置为 12，看是否显示正常
    self.form.ui.tripleSecSpinBox.setValue(12)
    print('* 当执行 self.form.ui.tripleSecSpinBox.setValue(12) 后，
ui.tripleSecSpinBox.value() => ' +
str( self.form.ui.tripleSecSpinBox.value() ) )
    self.assertEqual(self.form.ui.tripleSecSpinBox.value(), 11 )

    # tripleSecSpinBox 在界面中的取值范围为 0~11，将它的最小值设置为 -1，看是否显示正常
    self.form.ui.tripleSecSpinBox.setValue(-1)
    print('* 当执行 self.form.ui.tripleSecSpinBox.setValue(-1) 后，
ui.tripleSecSpinBox.value() => ' +
str( self.form.ui.tripleSecSpinBox.value() ) )
    self.assertEqual(self.form.ui.tripleSecSpinBox.value(), 0 )

    self.form.ui.tripleSecSpinBox.setValue(2)

    # 用鼠标左键单击“OK”按钮
    okWidget = self.form.ui.okBtn
    QTest.mouseClick(okWidget, Qt.LeftButton)
    self.assertEqual(self.form.getJiggers(), 2)
    print('*** testCase test_tripleSecSpinBox end ***')
'''

```

□□□□□□□□□□□□□□□□9-57□□□□



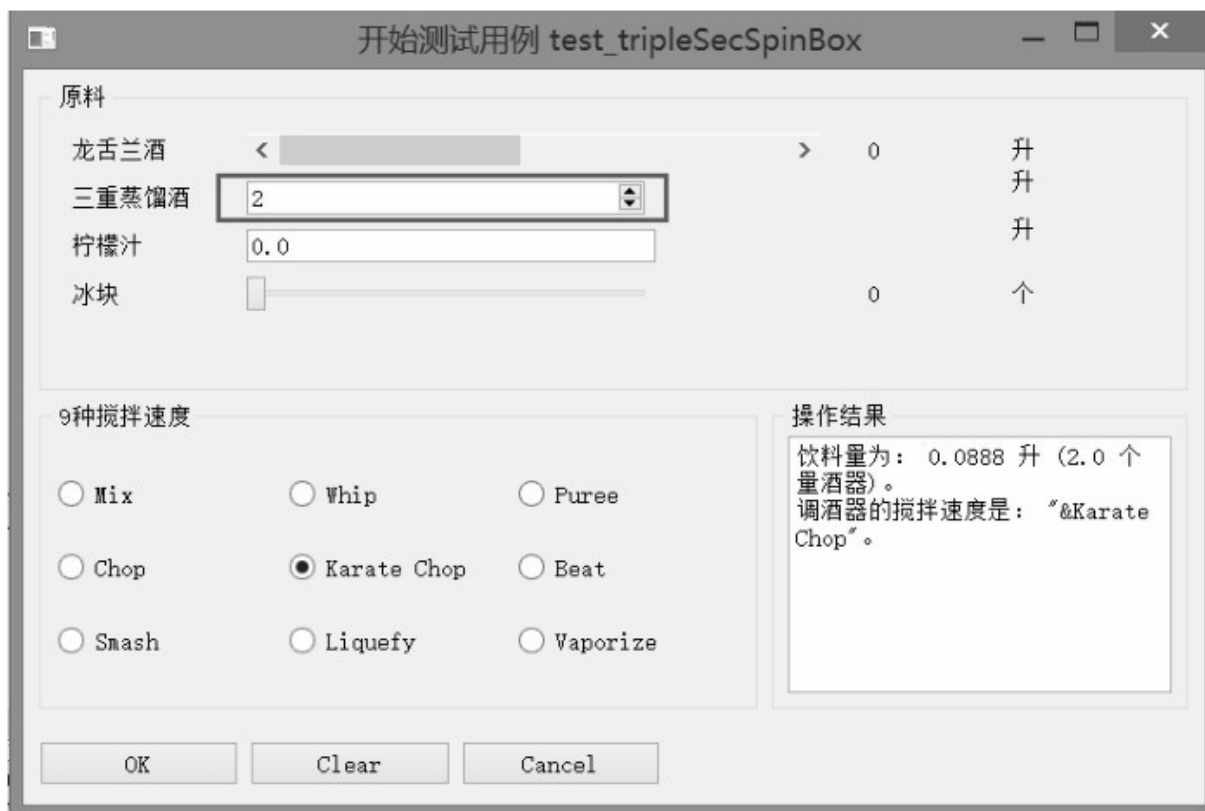


图9-57

## 6. 使用PyQt的QLineEdit

在QTest.keyClicks()方法中，我们使用limeJuiceLineEdit()方法来设置文本框的文本。在代码清单9-57中，我们使用QTest.keyClicks()方法来设置QtTest.QLineEdit.setText()方法来设置文本框的文本。

```

# 测试用例——测试柠檬汁单行文本框
def test_limeJuiceLineEdit(self):
    '''测试用例 test_limeJuiceLineEdit '''
    print('*** testCase test_limeJuiceLineEdit begin ***')
    self.form.setWindowTitle('开始测试用例 test_limeJuiceLineEdit ')

    '''测试修改 lineEdit 文本框控件的最大值、最小值
    测试它的最小值和最大值作为读者的练习
    '''
    self.setFormToZero()
    # 清除 lineEdit 文本框控件值，然后在 lineEdit 文本框控件中输入"3.5"
    self.form.ui.limeJuiceLineEdit.clear()
    QTest.keyClicks(self.form.ui.limeJuiceLineEdit, "3.5")

    # 用鼠标左键单击“OK”按钮
    okWidget = self.form.ui.okBtn
    QTest.mouseClick(okWidget, Qt.LeftButton)
    self.assertEqual(self.form.getJiggers() , 3.5)
    print('*** testCase test_limeJuiceLineEdit end ***')

```

□□□□□□□□□□□□□□9-58□□□

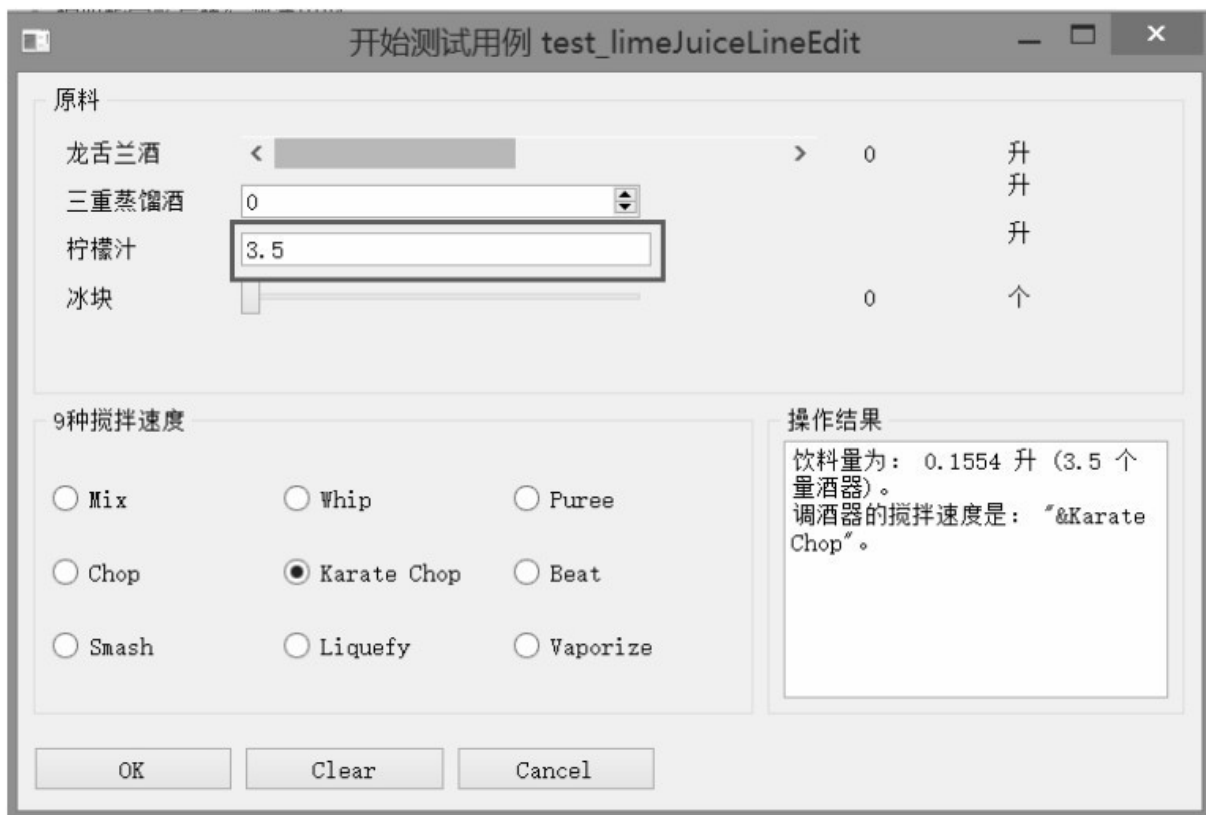


图9-58

## 7. PyQt QSlider

测试用例

```
# 测试用例——QSlider
def test_iceHorizontalSlider(self):
    """测试用例 test_iceHorizontalSlider"""
    print('*** test_iceHorizontalSlider begin ***')

    self.form.setWindowTitle(' 测试用例 test_iceHorizontalSlider ')

    """测试用例
    测试用例
    """
```

```

self.setFormToZero()
self.form.ui.iceHorizontalSlider.setValue(4)
# 单击“OK”按钮
okWidget=self.form.ui.okBtn
QTest.mouseClick(okWidget,Qt.LeftButton)
self.assertEqual(self.form.getJiggers(),4)
print('*** testCase test_iceHorizontalSlider end ***')

```

9-59



9-59

## 8. PyQt 的 QRadioButton

“9 种搅拌速度”的 9 种搅拌速度，单击“Mix”按钮，按下“Alt+M”组合键，单击“Mix”按钮。

□□□□□

```
# □□□□——□□□□□□□□□□
def test_blenderSpeedButtons(self):
    print('*** testCase test_blenderSpeedButtons begin
***')
    self.form.ui.speedButton1.click()
    self.assertEqual(self.form.getSpeedName(),"&Mix")
    self.form.ui.speedButton2.click()
    self.assertEqual(self.form.getSpeedName(),"&Whip")
    self.form.ui.speedButton3.click()
    self.assertEqual(self.form.getSpeedName(),"&Puree")
    self.form.ui.speedButton4.click()
    self.assertEqual(self.form.getSpeedName(),"&Chop")
    self.form.ui.speedButton5.click()
    self.assertEqual(self.form.getSpeedName(),"&Karate
Chop")
    self.form.ui.speedButton6.click()
    self.assertEqual(self.form.getSpeedName(),"&Beat")
    self.form.ui.speedButton7.click()
    self.assertEqual(self.form.getSpeedName(),"&Smash")
    self.form.ui.speedButton8.click()
    self.assertEqual(self.form.getSpeedName(),"&Liquefy")
    self.form.ui.speedButton9.click()
    self.assertEqual(self.form.getSpeedName(),"&Vaporize"
)
    print('*** testCase test_blenderSpeedButtons end ***')
□□□□□□□□□□□□□□□□9-60□□□□
```



图9-60

## 9.7.4 单元测试

下面通过编写MatrixWinTest.py文件来测试MatrixWin.py文件中的函数。

编写unittest.py文件

```
import unittest
```

**1.**编写测试类

编写测试类，继承unittest.TestCase

```
if __name__ == "__main__":
    unittest.main()
```

运行测试，生成报告文件 reportLog.txt

```
python MatrixWinTest.py --reportLog.txt 2>&1
```

## 2. unittest 模組

unittest 模組是 Python 標準庫中的一個模組，用於測試 Python 程式。

```
if __name__ == "__main__":
    suite=unittest.TestSuite()
    suite.addTest(MatrixWinTest("test_defaults"))
    suite.addTest(MatrixWinTest("test_moveScrollBar"))
    suite.addTest(MatrixWinTest("test_tripleSecSpinBox")
)
    suite.addTest(MatrixWinTest("test_limeJuiceLineEdit")
)
    suite.addTest(MatrixWinTest("test_iceHorizontalSlider
"))
    suite.addTest(MatrixWinTest("test_liters"))
    suite.addTest(MatrixWinTest("test_blenderSpeedButt
ons"))
    runner=unittest.TextTestRunner()
    runner.run(suite)
```

### 9.7.5 HTMLTestRunner

HTMLTestRunner 是 unittest 模組的一個子類別，用於生成 HTML 格式的測試報告。

HTMLTestRunner 是 Python 標準庫中的一個模組。

HTMLTestRunner 是 Python 標準庫中的一個子類別，用於生成 HTML 格式的測試報告。

1. HTMLTestRunner.py 檔案

<http://tungwaiyip.info/software/HTMLTestRunner.html> 9-61

☐☐☐

HTMLTestRunner☐☐☐☐Python 2☐☐☐☐☐HTMLTestRunner☐☐  
☐ ☐ ☐ Python 3 ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐ ☐  
<http://www.cnblogs.com/sgtb/p/4169732.html> ☐ ☐ ☐  
HTMLTestRunner.py☐☐☐☐☐☐Python 3☐☐☐



☐9-61

☐ ☐ Python 3 ☐ ☐ ☐ HTMLTestRunner.py ☐ ☐ ☐ ☐  
PyQt5/Chapter03/testCase☐☐☐☐☐☐☐☐☐☐  
☐2☐☐☐☐☐☐☐HTMLTestRunner.py☐☐☐☐Python3/lib☐☐☐☐☐☐☐  
☐☐☐☐☐E:\installed\_software\python35\Lib☐☐☐  
☐3☐☐☐☐☐☐☐



PyQt5/Chapter03/testCase/RunTestCase.py  
HTMLTestRunner

```
import unittest
import HTMLTestRunner
import time
from MatrixWinTest import MatrixWinTest

if __name__ == "__main__":

    now = time.strftime("%Y-%m-%d-%H_%M_%S",
time.localtime(time.time()))
    print( now )
    testunit = unittest.TestSuite()
    testunit.addTest(unittest.makeSuite(MatrixWinTest ))

    htmlFile = ".\\"+now+"HTMLtemplate.html"
    print( 'htmlFile='+ htmlFile)
    fp = open(htmlFile,'wb')
    runner = HTMLTestRunner.HTMLTestRunner(
        stream=fp,
        title=u"PyQt5 测试报告",
        description=u"用例测试情况")
    runner.run(testunit)
    fp.close()
```

{ -  
}HTMLtemplate.html 2017-05-26-  
20\_22\_06HTMLtemplate.html9-62  
pass

PyQt5测试报告

Start Time: 2017-05-28 16:27:51

Duration: 0:00:35.213974

Status: Pass 7

用例测试情况

Show Summary Failed All

Test Group/Test case	Count	Pass	Fail	Error	View
MatrixWinTest.MatrixWinTest	7	7	0	0	Detail
test_blenderSpeedButtons			pass		
test_defaults: 测试GUI处于默认状态			pass		
test_iceHorizontalSlider: 测试用例 test_iceHorizontalSlider			pass		
test_limeJuiceLineEdit: 测试用例 test_limeJuiceLineEdit			pass		
test_liters: 测试用例 test_liters			pass		
test_moveScrollBar: 测试用例test_moveScrollBar			pass		
test_tripleSecSpinBox: 测试用例 test_tripleSecSpinBox			pass		
Total	7	7	0	0	

# 第10章 PyQt 5 网络编程

## 10.1 网络编程

Python 提供了多种模块用于 HTTP 网络编程，本章将介绍使用 urllib 模块进行网络编程。

### 10.1.1 网络编程

Python 提供了多种模块用于 HTTP 网络编程，本章将介绍使用 urllib 模块进行网络编程。HTML 解析使用 XPath 和 BeautifulSoup 模块，HTML 解析使用 BeautifulSoup 模块。API 接口使用 BeautifulSoup 模块解析 HTML 文档，API 接口使用 BeautifulSoup 模块解析 HTML 文档。API 接口使用 BeautifulSoup 模块解析 HTML 文档，API 接口使用 BeautifulSoup 模块解析 HTML 文档。

Requests 是 Python 中一个非常流行的 HTTP 库，可以通过 pip 安装。

pip install requests

Requests 支持 json 数据格式，JSON 是 Python 中一种常用的数据格式。Requests 支持 json 数据格式，JSON 是 Python 中一种常用的数据格式。



图10-1

## 10.1.2 天气数据API

通过<http://www.weather.com.cn/data/sk/城市代码.html>

获取城市天气数据10-1

图10-1

城市名称	城市代码
北京	101010100
天津	101030100
上海	101020100

图10-2

图10-2



PyQt5/Chapter10/example1/getWeatherInfo.py

```
import requests
rep=requests.get('http://www.weather.com.cn/data/sk/101010100.html')
rep.encoding='utf-8'
print('城市: %s' % rep.json() ) print('天气: %s' %
rep.json()['weatherinfo']['city'])
print('温度: %s' % rep.json()['weatherinfo']['WD'])
print('湿度: %s' % rep.json()['weatherinfo']['temp']+ "
")
print('风速: %s' % rep.json()['weatherinfo']['WS'])
print('风向: %s' % rep.json()['weatherinfo']['SD'])
```

HTTP GET 返回的响应文本

Response Text 返回的响应文本是JSON格式

utf-8 编码的文本

```
{'weatherinfo': {'city': '北京', 'cityid':
'101010100', 'temp': '18', 'WD': '晴', 'WS': '1级', 'SD':
'17%', 'WSE': '1', 'time': '17:05', 'isRadar': '1', 'Radar':
'JC_RADAR_AZ9010_JB', 'njd': '北京', 'qy': '1011', 'rain': '0'}}
```

城市: 北京

天气: 晴

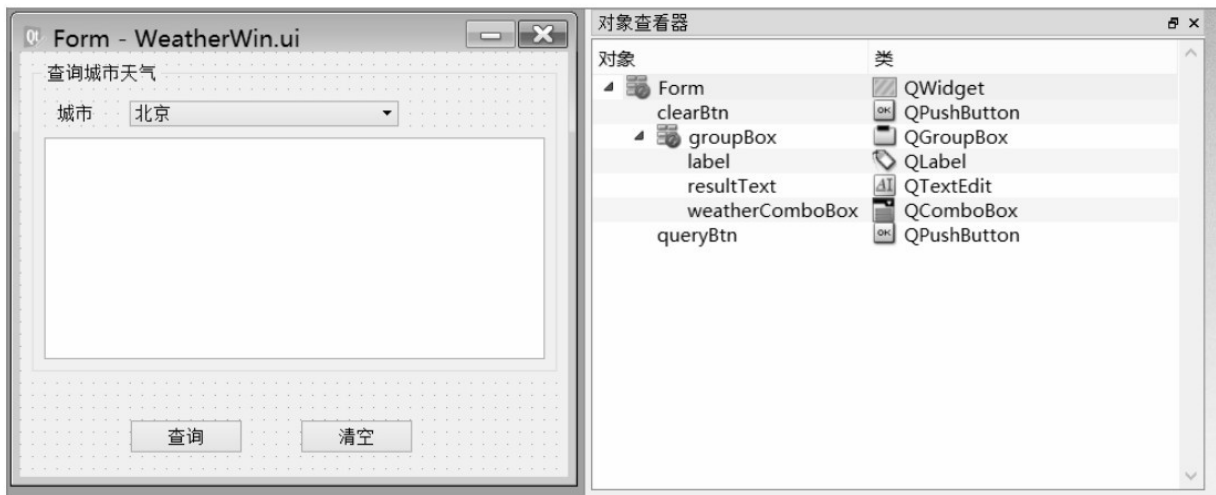
温度: 18 °C

湿度: 17%

□□: 17%

### 10.1.3 □□□□

□□ Qt Designer□□□□□□□□□□ 10-2□□ 10-3□□□□□□□□□  
PyQt5/Chapter10/example1/WeatherWin.ui□



□10-2



□10-3

图 10-3 显示了 WeatherWin.ui 文件中的 UI 设计。图中包含一个名为 `weatherComboBox` 的下拉列表框，一个名为 `resultText` 的文本框，以及两个名为 `queryBtn` 和 `clearBtn` 的按钮。图中还显示了 `queryWeather()` 和 `clearResult()` 函数的调用。

“图 10-3”显示了 WeatherWin.ui 文件中的 UI 设计。图中包含一个名为 `weatherComboBox` 的下拉列表框，一个名为 `resultText` 的文本框，以及两个名为 `queryBtn` 和 `clearBtn` 的按钮。图中还显示了 `queryWeather()` 和 `clearResult()` 函数的调用。

图 10-3 显示了 WeatherWin.ui 文件中的 UI 设计。

图 10-3

控件类型	控件名称	作用
QComboBox	weatherComboBox	显示城市下拉列表框，添加了 3 个条目：北京、天津和上海。
QTextEdit	resultText	显示查询天气的结果
QPushButton	queryBtn	查询天气信息，连接 <code>queryWeather</code> 函数进行绑定，触发 <code>clicked</code> 信号
QPushButton	clearBtn	清空查询结果，连接 <code>clearResult</code> 函数进行绑定，触发 <code>clicked</code> 信号

### 10.1.4 生成 WeatherWin.py

使用 `pyuic5` 工具生成 `WeatherWin.py` 文件。Python 代码如下：

`pyuic5 -o WeatherWin.py WeatherWin.ui`

该命令将在 `PyQt5/Chapter10/example1/WeatherWin.py` 目录下生成 `WeatherWin.py` 文件。

代码如下：

```
from PyQt5 import QtCore, QtGui, QtWidgets
```



```

class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(450, 347)
        self.groupBox = QtWidgets.QGroupBox(Form)
        self.groupBox.setGeometry(QtCore.QRect(10, 10, 431, 251))
        self.groupBox.setObjectName("groupBox")
        self.weatherComboBox = QtWidgets.QComboBox(self.groupBox)
        self.weatherComboBox.setGeometry(QtCore.QRect(80, 30, 221, 21))
        self.weatherComboBox.setObjectName("weatherComboBox")
        self.weatherComboBox.addItem("")
        self.weatherComboBox.addItem("")
        self.weatherComboBox.addItem("")
        self.resultText = QtWidgets.QTextEdit(self.groupBox)
        self.resultText.setGeometry(QtCore.QRect(10, 60, 411, 181))
        self.resultText.setObjectName("resultText")
        self.label = QtWidgets.QLabel(self.groupBox)
        self.label.setGeometry(QtCore.QRect(20, 30, 72, 21))
        self.label.setObjectName("label")
        self.okButton = QtWidgets.QPushButton(Form)
        self.okButton.setGeometry(QtCore.QRect(90, 300, 93, 28))
        self.okButton.setObjectName("okButton")
        self.clearBtn = QtWidgets.QPushButton(Form)
        self.clearBtn.setGeometry(QtCore.QRect(230, 300, 93, 28))
        self.clearBtn.setObjectName("clearBtn")

        self.retranslateUi(Form)
        self.clearBtn.clicked.connect(Form.clearResult)
        self.okButton.clicked.connect(Form.queryWeather)
        QtCore.QMetaObject.connectSlotsByName(Form)

    def retranslateUi(self, Form):
        _translate = QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form", "Form"))
        self.groupBox.setTitle(_translate("Form", "查询城市天气"))
        self.weatherComboBox.setItemText(0, _translate("Form", "北京"))
        self.weatherComboBox.setItemText(1, _translate("Form", "天津"))
        self.weatherComboBox.setItemText(2, _translate("Form", "上海"))
        self.label.setText(_translate("Form", "城市"))
        self.okButton.setText(_translate("Form", "查询"))
        self.clearBtn.setText(_translate("Form", "清空"))

```

### 10.1.5 调用天气API

MainWindow 包含 Ui\_Form 类，该类是 QMainWindow 的子类，用于定义主窗口的 UI 布局。

在 PyQt5/Chapter10/example1/CallWeatherWin.py 文件中，我们调用了 queryWeather() 和 clearResult() 方法。在 WeatherWin.ui 文件中，我们调用了 queryBtn 和 clearBtn 的 clicked 信号。

```

import sys
from PyQt5.QtWidgets import QApplication , QMainWindow
from WeatherWin import Ui_Form
import requests

class MainWindow(QMainWindow ):
    def __init__(self, parent=None):
        super(MainWindow, self).__init__(parent)
        self.ui = Ui_Form()
        self.ui.setupUi(self)

    def queryWeather(self):
        print('* queryWeather ')
        cityName = self.ui.weatherComboBox.currentText()
        cityCode = self.transCityName(cityName)

        rep = requests.get('http://www.weather.com.cn/data/sk/' +
cityCode + '.html')
        rep.encoding = 'utf-8'
        print( rep.json() )

        msg1 = '城市: %s' % rep.json()['weatherinfo']['city'] + '\n'
        msg2 = '风向: %s' % rep.json()['weatherinfo']['WD'] + '\n'
        msg3 = '温度: %s' % rep.json()['weatherinfo']['temp'] + ' 度' +
'\n'
        msg4 = '风力: %s' % rep.json()['weatherinfo']['WS'] + '\n'
        msg5 = '湿度: %s' % rep.json()['weatherinfo']['SD'] + '\n'
        result = msg1 + msg2 + msg3 + msg4 + msg5
        self.ui.resultText.setText(result)

    def transCityName(self ,cityName):
        cityCode = ''

```

```

        if cityName == '北京' :
            cityCode = '101010100'
        elif cityName == '天津' :
            cityCode = '101030100'
        elif cityName == '上海' :
            cityCode = '101020100'

    return cityCode

def clearResult(self):
    print('* clearResult ')
    self.ui.resultText.clear()

if __name__=="__main__":
    app = QApplication(sys.argv)
    win = MainWindow()
    win.show()
    sys.exit(app.exec_())

```

10-4

Form

查询城市天气

城市: 北京

城市: 北京  
 风向: 东南风  
 温度: 18 度  
 风力: 1级  
 湿度: 17%

查询 清空

## 10.2 複利

### 10.2.1 複利計算

複利とは、元金に利息を加えて、その合計が次の元金となる計算方法のことである。例えば、元金1000円に年利5%の複利で2年間放置した場合、2年後の元金はいくらになるかを計算する。

複利計算の式は、 $S = P(1+i)^n$  である。ここで、 $S$  は元金、 $P$  は元金、 $i$  は年利、 $n$  は年数である。

例えば、元金20000円に年利5%の複利で2年間放置した場合、2年後の元金はいくらになるかを計算する。

複利計算の式は、 $S = P(1+i)^n$  である。ここで、 $S$  は元金、 $P$  は元金、 $i$  は年利、 $n$  は年数である。

$$\begin{aligned} 20000 \times (1+5\%)^2 &= 20000 \times (1+5\%) \times (1+5\%) \\ &= 20000 \times 1.05 \times 1.05 \\ &= 22050 \text{円} \end{aligned}$$

### 10.2.2 複利計算

本プログラムはPyQt5/Chapter10/example2/interest.pyで実装されている。

```
from __future__ import division
import sys
from PyQt5.QtWidgets import (QApplication, QComboBox, QDialog,
                              QDoubleSpinBox, QGridLayout, QLabel)

class Form(QDialog):

    def __init__(self, parent=None):
        super(Form, self).__init__(parent)

        principalLabel = QLabel("Principal:")
        self.principalSpinBox = QDoubleSpinBox()
        self.principalSpinBox.setRange(1, 1000000000)
        self.principalSpinBox.setValue(1000)
        self.principalSpinBox.setPrefix("RMB ")
        rateLabel = QLabel("Rate:")
        self.rateSpinBox = QDoubleSpinBox()
        self.rateSpinBox.setRange(1, 100)
        self.rateSpinBox.setValue(5)
        self.rateSpinBox.setSuffix(" %")
        yearsLabel = QLabel("Years:")
        self.yearsComboBox = QComboBox()
```

```

        self.yearsComboBox.addItem("1 year")
        self.yearsComboBox.addItems(["{0} years".format(x)
                                     for x in range(2, 31)])
        amountLabel = QLabel("Amount")
        self.amountLabel = QLabel()

        grid = QGridLayout()
        grid.addWidget(principalLabel, 0, 0)
        grid.addWidget(self.principalSpinBox, 0, 1)
        grid.addWidget(rateLabel, 1, 0)
        grid.addWidget(self.rateSpinBox, 1, 1)
        grid.addWidget(yearsLabel, 2, 0)
        grid.addWidget(self.yearsComboBox, 2, 1)
        grid.addWidget(amountLabel, 3, 0)
        grid.addWidget(self.amountLabel, 3, 1)
        self.setLayout(grid)

        self.principalSpinBox.valueChanged.connect(self.updateUi)
        self.rateSpinBox.valueChanged.connect(self.updateUi)
        self.yearsComboBox.currentIndexChanged.connect(self.updateUi)

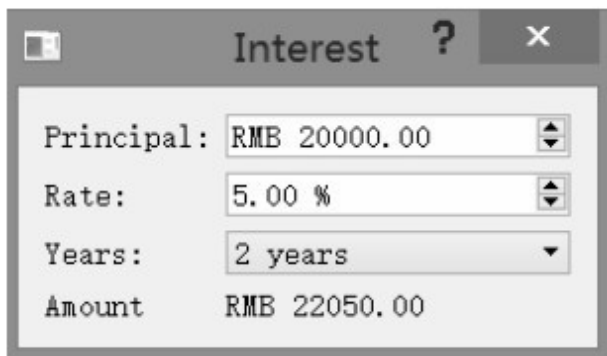
        self.setWindowTitle("Interest")
        self.updateUi()

    def updateUi(self):
        principal = self.principalSpinBox.value()
        rate = self.rateSpinBox.value()
        years = self.yearsComboBox.currentIndex() + 1
        amount = principal * ((1 + (rate / 100.0)) ** years)
        self.amountLabel.setText("RMB {0:.2f}".format(amount))

if __name__ == "__main__":
    app = QApplication(sys.argv)
    form = Form()
    form.show()
    sys.exit(app.exec_())

```

假设本金为10-5元，利率为20000元，期限为5%，则  
本息和为22050元。



Interest ? x

Principal: RMB 20000.00

Rate: 5.00 %

Years: 2 years

Amount RMB 22050.00

图10-5

## 10.3 两行代码

本节将介绍一种两行代码搞定计数的方法，该方法简单易学，且无需任何第三方服务，即可实现计数的功能。

访问<http://busuanzi.ibruce.info/>，如图10-6所示。



图10-6



```

<script>
    <script async src="//dn-lbstatics.qbox.me/busuanzi/2.3/busuanzi.pure.mini.js">
</script>
    <span id="busuanzi_container_site_pv">
    <span id="busuanzi_value_site_pv"></span>
    <br>
    <span id="busuanzi_value_page_pv"></span>
    </span>
    <a href="http://www.cnblogs.com/wangshuo1/">
    <span id="busuanzi_value_page_pv">10-7</span>

```



图10-7

在PyQt5/Chapter10/example3/openweb.py中添加如下代码

```

from PyQt5.QtWebEngineWidgets import
QWebEngineView
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
class WebView(QWebEngineView):

```

```
def __init__(self ):
    super(WebView,self).__init__()
    url='http://www.cnblogs.com/wangshuo1/p/670763
1.html'
    self.load( QUrl( url ) )
    self.show()
    # 每隔5秒刷新一次
    QTimer.singleShot(1000*5 ,self.close)
    webView = QWebEngineView(self)
    QTimer.singleShot(1000*5 ,self.close)
    webView.load(QUrl.fromLocalFile('PyQt5/Chapter10/example3/CallOpenWeb.py'))
    webView.show()

import time
import os
if __name__ == '__main__':
    for i in range(5):
        os.system("python openweb.py")
        print("已经运行了 %d 次" ,i)
        time.sleep(10)
    QTimer.singleShot(1000*5 ,self.close)
    QTimer.singleShot(1000*5 ,self.close)
```



图10-8

通过调用 `CallOpenWeb.py` 打开网页 <http://www.cnblogs.com/wangshuo1/p/6707631.html>，  
 统计118次访问，10-9

本站总访问量1308次  
 本文总阅读量118次

图10-9

通过调用 `CallOpenWeb.py` 打开网页 <http://www.cnblogs.com/wangshuo1/p/6707631.html>，  
 统计118次访问，10-10 统计5次访问，  
 $118 + 5 + 1 = 124$

本站总访问量1315次

本文总阅读量124次

□10-10

# 11 PyQt 5

本章主要介绍 PyQt 5 的相关知识，包括 PyQt 5 的安装、  
使用等。

本章主要介绍 Qt 的相关知识，包括 Qt 的安装、使用 WPS  
Office、Photoshop 等软件，以及 PyQt 与 Qt 的关系等。  
本章主要介绍

本章主要介绍 PyQt 5 的相关知识，包括 IT 行业、  
PyQt 5 的安装、使用等。本章主要介绍 PyQt 5 的  
安装、使用等。本章主要介绍 PyQt 5 的安装、  
使用等。本章主要介绍 PyQt 5 的安装、使用等。  
本章主要介绍 PyQt 5 的安装、使用等。本章  
主要介绍 PyQt 5 的安装、使用等。本章主要  
介绍 PyQt 5 的安装、使用等。本章主要介绍  
PyQt 5 的 GUI 设计。

## 11.1

本章主要介绍 Qt Designer 的相关知识，包括 Scroll Area  
Widget、Tab 等。

本章主要介绍 Widget 的相关知识，包括 11-1 图所示的 QWidget  
参数树“widget\_parameter\_tree”。



图11-1

在Tab页中添加QWebEngineView，并设置其父部件为QWidget。

在QWebEngineView的属性面板中，将“background-color”属性设置为“#FFFFFF”。

图11-2



④ 单击 `widget_parameter_tree` 树形结构中的 `verticalLayout_3` 节点，将其名称从 `verticalLayout_3` 更改为 `scrollArea`。

单击 `verticalLayout_3` 节点，将其名称从 `verticalLayout_3` 更改为 `scrollArea`。

单击 `verticalLayout_3` 节点，将其名称从 `verticalLayout_3` 更改为 `scrollArea`。



图11-3

单击 `scrollArea` 节点，将其名称从 `scrollArea` 更改为 `scrollArea`。



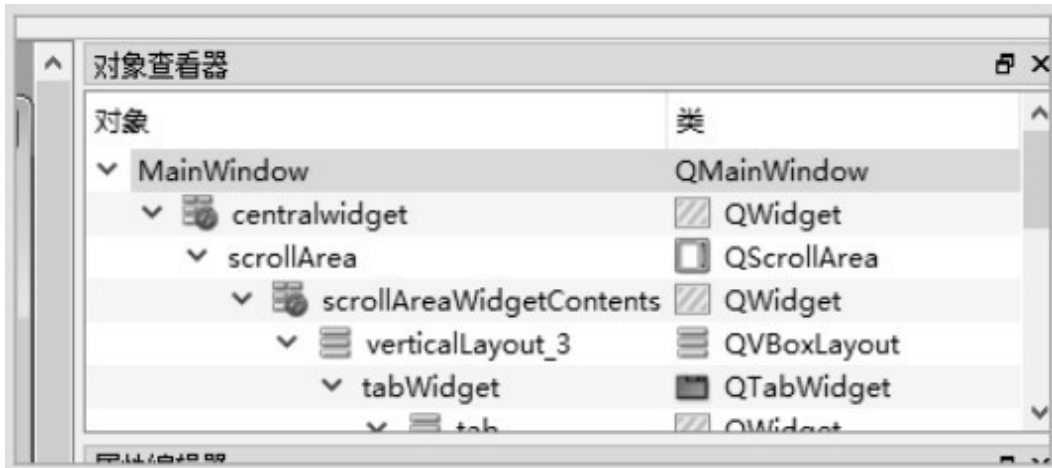


图11-4

在 widget\_parameter\_tree 中，将 scrollAreaWidgetContents 的 width 属性值从 100 修改为 500，并将 scrollArea 的 scrollAreaWidgetContents 属性值从 "scrollAreaWidgetContents" 修改为 "verticalLayout\_3"。修改后的效果如图 11-5 所示。

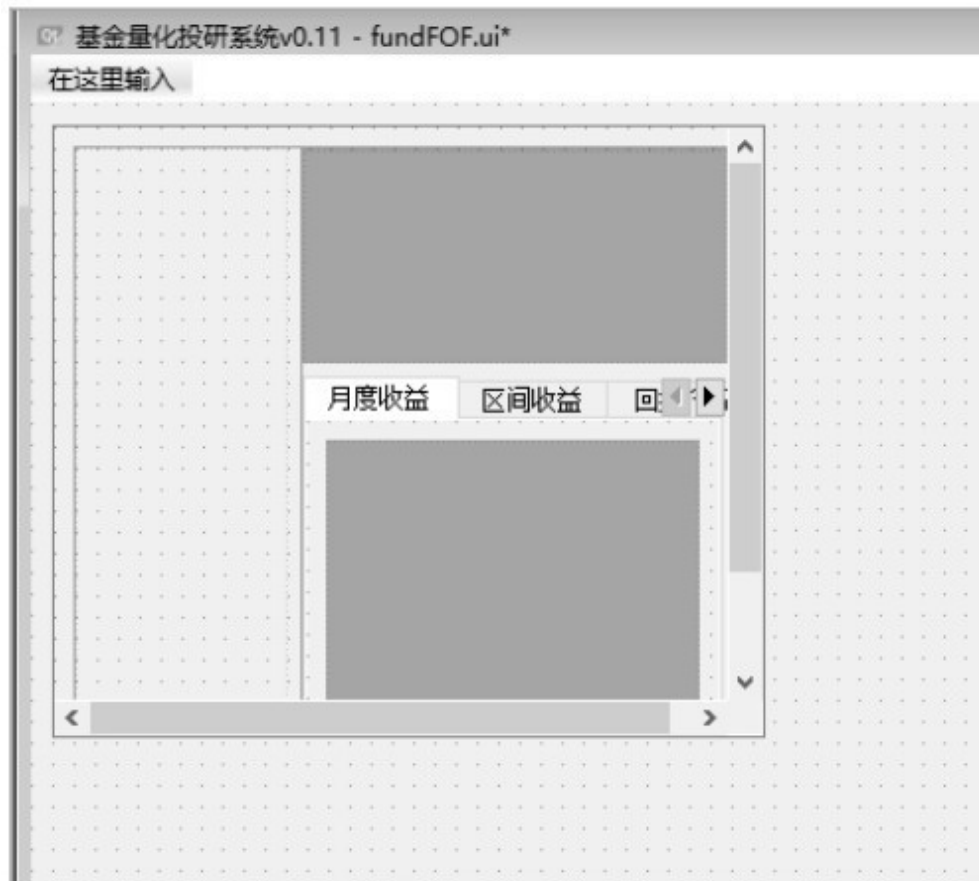


图11-5

## 11.2 数据展示

在数据展示页面，主要展示基金收益数据，包括基金名称、基金代码、基金类型、基金规模、基金收益等。

在数据展示页面，主要展示基金收益数据，包括基金名称、基金代码、基金类型、基金规模、基金收益等。

在数据展示页面，主要展示基金收益数据，包括基金名称、基金代码、基金类型、基金规模、基金收益等。

“”→“”图11-6



□11-6

```

    widget_parameter_tree[0] = scrollArea
    widget_parameter_tree[1] = ...

```



PyQt 5 ParameterTree  
pyqtgraph QTreeWidget QTreeWidget

11

基金量化投资系统v0.11

### 千石资本-和聚光明1号资产管理计划基本信息

参数	数值
<b>基本收益信息</b>	
今年收益	-2.45 %
累计收益	-2.72 %
最近净值日期	2017-02-03
最新净值	0.973
累计净值	0.973
<b>基本产品信息</b>	
开放日	0个月
累计收益	封闭期后，每个自然季度
认购起点	100 万(人民币)
追加起点	10 万(人民币)
认购费率	1 %
赎回费率	0
持仓线	95 %
止损线	90 %
<b>关联信息</b>	
投资顾问	和聚投资
业绩报酬	20 %
基金管理人	千石资本
管理人管理费	0
基金托管人	光大银行
证券经纪商	未设
期货经纪商	未设
<b>其他信息</b>	
成立日期	2015-08-04
是否分级	否
产品类型	公募专属
投资策略	股票策略
子策略	股票多头

### 和聚光明1号 净值走势图

和聚光明1号  
沪深300

1.038 和聚光明1号  
0.7789203 沪深300  
May 13, 2016

### 和聚光明1号 收益分布图

月收益  
区间收益  
创新情况

今年以来  
最近一个月  
最近三个月  
最近半年  
最近一年

14.2 沪深300  
7.03 同类平均  
-5.84 本基金

□11-8

11-8 ParameterTree QWidget  
QWebEngineView 300  
ParameterTree

11-8

PyQt5/Chapter11/fundFOF.py

```

def __init__(self, parent=None):
    """
    Constructor

    @param parent reference to the parent widget
    @type QWidget
    """

    super(MainWindow, self).__init__(parent)

```

```

self.setupUi(self)
self.plotly_pyqt5 = Plotly_PyQt5()

'''手动调整窗口控件的大小，使之看起来更美观'''
self.widget_parameter_tree.setMaximumWidth(300)
self.widget_parameter_tree.setMinimumWidth(200)
self.QWebEngineView_ProductVsHs300.setMinimumHeight(500)
self.tabWidget.setMinimumHeight(400)

```

## ParameterTree

'''显示 parametertree，这里通过布局管理器把 ParameterTree 间接地嵌套到 Widget 窗口中'''

```

from mypyqtgraph import p
from pyqtgraph.parametertree import ParameterTree

t = ParameterTree()
t.setParameters(p, showTop=False)
t.setHeaderLabels(["参数", "数值"])
# t.setWindowTitle('pyqtgraph example: Parameter Tree')
layout = QtGui.QGridLayout()
self.widget_parameter_tree.setLayout(layout)
layout.addWidget(
    QtGui.QLabel("千石资本-和聚光明 1 号资产管理计划基本信息"), 0, 0, 1, 1)
layout.addWidget(t)

```

## from mypyqtgraph import p

from mypyqtgraph import p

```

t=ParameterTree()
t.setParameters(p,showTop=False)
mypyqtgraph
PyQt5/Chapter11/mypyqtgraph.py

```

```

# 创建参数树的数据
params = [
    {'name': '基本收益信息', 'type': 'group', 'children': [
        {'name': '今年收益', 'type': 'float', 'value': -2.45, 'siPrefix':
True, 'suffix': '%'},
        {'name': '累计收益', 'type': 'float', 'value': -2.72, 'step': 0.1,
'siPrefix': True, 'suffix': '%'},
        {'name': '最近净值日期', 'type': 'str', 'value': "2017-02-03"},
        {'name': '最新净值', 'type': 'float', 'value': 0.9728, 'step':
0.01},
        {'name': '累计净值', 'type': 'float', 'value': 0.9728, 'step':
0.01},
    ]},
    .....,
## 创建参数树
p = Parameter.create(name='params', type='group', children=params)

```

图11-9

参数	数值	
基本收益信息		
今年收益	-2.45 %	
累计收益	-2.72 %	
最近净值日期	2017-02-03	
最新净值	0.993	
累计净值	0.973	

图11-9

```

        'type': 'group'
        'type': 'str'
lineEdit
        'type': 'float'
doubleSpinBox
        " "
        value
        change

```

```

## 若树里面的任何内容发生变化，则输出这些变化
def change(param, changes):
    print("tree changes:")
    for param, change, data in changes:
        path = p.childPath(param)
        if path is not None:
            childName = '.'.join(path)
        else:
            childName = param.name()
        print(' parameter: %s' % childName)
        print(' change:    %s' % change)
        print(' data:      %s' % str(data))
        print(' -----')

p.sigTreeStateChanged.connect(change)

```

```

        value
        valueChanging
def valueChanging(param,value):
    print("Value changing (not finalized):",param,value)
    # Too lazy for recursion:
    for child in p.children():
        child.sigValueChanging.connect(valueChanging)
        for ch2 in child.children():
            ch2.sigValueChanging.connect(valueChanging)
        ParameterTree
pyqtgraph
ParameterTree
import pyqtgraph.examples

```



```

pyqtgraph.examples.run()
#####
'''#####'''
    self.QWebEngineView_ProductVsHs300.load(
        QUrl.fromLocalFile(self.plotly_pyqt5.get_plotly_path_pro
duct_vs_hs300()))
    self.QWebEngineView_LagestBack.load(QUrl.fromLoc
alFile(self.plotly_py qt5.get_plotly_path_lagest_back()))
    self.QWebEngineView_PeriodReturn.load(QUrl.fromLo
calFile(self.plotly_pyqt5.get_plotly_path_period_return()))
    self.QWebEngineview_MonthReturn.load(QUrl.fromLo
calFile(self.plotly_p
yqt5.get_plotly_path_month_return()))
    ##### QWebEngineView #####HTML
    ##### plotly_pyqt5.get_plotly_path_product_vs_hs300()#####
    plotly#####HTML#####
    ##### plotly_pyqt5.get_plotly_path_product_vs_hs300()#####
    #####PyQt5/Chapter11/Plotly_PyQt5.py#####

```

```

def get_plotly_path_product_vs_hs300(self,
file_name='product_vs_hs300.html'):
    path_plotly = self.path_dir_plotly_html + os.sep + file_name

    data = pd.read_excel(r'data\和聚光明1号_hs300_merge.xlsx',
index_col=[0] )
    data.rename_axis(lambda x: pd.to_datetime(x), inplace=True)
    data.dropna(inplace=True)

    data = [
        go.Scatter(
            x=data.index, # assign x as the dataframe column 'x'
            y=data.cumulative_nav,
            name='和聚光明1号'
        ),

```

```

        go.Scatter(
            x=data.index, # assign x as the dataframe column 'x'
            y=data.close,
            name='沪深300'
        )
    ]
    pyof.plot(data, filename=path_plotly, auto_open=False)
    return path_plotly

```

#####HTML#####

path\_plotly=self.path\_dir\_plotly\_html + os.sep +  
file\_name

data=pd.read\_excel(r'data\ 和聚光明1号  
\_hs300\_merge.xlsx',  
index\_col=[0])

#####data#####data#####  
#####data  
#####10#####

data.rename\_axis(lambda x: pd.to\_datetime(x),inplace=True)

```
data.dropna(inplace=True)
```

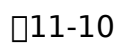
```
pyof.plot(auto_open=False)
filename=path_plotlyHTMLpath_plotly
```

```
data = [
    go.Scatter(
        x=data.index, # assign x as the dataframe column 'x'
        y=data.cumulative_nav,
        name='和聚光明 1 号'
    ),
    go.Scatter(
        x=data.index, # assign x as the dataframe column 'x'
        y=data.close,
        name='沪深 300'
    )
]
pyof.plot(data, filename=path_plotly, auto_open=False)
return path_plotly
```

```
mini
```

## 11.3.2

11-10



11-11

	最小范围	最大范围
收益	0.00	0.00
最大回撤	0.70	1.00
Sharp比	0.00	10.00

☐ 管理期货

☐ 事件驱动

☒ 债券策略

☒ 宏观策略

☐ 组合基金

开始

11-11

GUI

PyQt5/Chapter11/comboination.ui

“”

```
@pyqtSlot()
def on_pushButton_start_combination_clicked(self):
    """
    """

    strategy_list=self.check_check_box()
    self.check_check_box()
    strategy_list=[]
    def check_check_box(self):
        strategy_list=[]
```

```

        for each_checkBox in [self.checkBox_bond,self.checkBox_
        combination_fund,self.checkBox_compound,self.checkBox_
        ox_event,self.checkBox_future_manage,self.checkBox_macro,self.checkBox_relati
        ve_fund,self.checkBox_others,self.checkBox_others]:
            if each_checkBox.isChecked():
                strategy_list.append(each_checkBox.text())
        return strategy_list

    def check_check_box(self):
        strategy_list = []
        if len(strategy_list) > 3:
            QMessageBox.information(self, "提示", "策略列表已满，请删除部分策略")
            return None
        if len(strategy_list) == 0:
            QMessageBox.information(self, "提示", "策略列表为空，请添加策略")
            return None
        strategy_list = self.check_check_box()
        if len(strategy_list) > 3:
            QMessageBox.information(self, "提示", "策略列表已满，请删除部分策略")
            return None
        if len(strategy_list) == 0:
            QMessageBox.information(self, "提示", "策略列表为空，请添加策略")
            return None
        self.QWebEngineview_Combination_monte_markovitz.setMinimumHeight(800)

```

```

        self.QWebEngineview_Combination_Pie.setMinimumHeight(400)
        self.QWebEngineview_Combination_Table.setMinimumHeight(400)
        self.QWebEngineview_Combination_Versus.setMinimumHeight(700)
        print('min:',self.doubleSpinBox_returns_min.text())
        print('max:',self.doubleSpinBox_returns_max.text())
        print('min:',self.doubleSpinBox_maxdrawdown_min.text())
        print('max:',self.doubleSpinBox_maxdrawdown_max.text())
        print('sharp_min:',self.doubleSpinBox_sharp_min.text())
        print('sharp_max:',self.doubleSpinBox_sharp_max.text())
        print('3 1 0.4 0.2 0.4')
        print('')
        df=pd.read_excel(r'data\1.xlsx',index_col=[0])
        w=[0.4,0.2,0.4]
        df['']=(df * w).sum(axis=1)
        print('')
        self.QWebEngineview_Combination_monte_markovitz.load(
            QUrl.fromLocalFile(self.plotly_pyqt5.get_plotly_path_monte_markovitz( monte_count=600)))

```

```
self.QWebEngineview_Combination_Pie.load(
    QUrl.fromLocalFile(self.plotly_pyqt5.get_plotly_path_co
mbination_pie( df=df,w=w)))
self.QWebEngineview_Combination_Versus.load(
    QUrl.fromLocalFile(self.plotly_pyqt5.get_plotly_path_co
mbination_vers us(df=df,w=w)))
self.QWebEngineview_Combination_Table.load(
    QUrl.fromLocalFile(self.plotly_pyqt5.get_plotly_path_co
mbination_tabl e(df=df,w=w)))
    □          □          □          □          □          □          □
self.plotly_pyqt5.get_plotly_path_monte_markovitz □□□□□□□□
□□□□□□□□□□□□
    □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□sharp□□
```



```

def get_plotly_path_monte_markovitz(self,
file_name='monte_markovitz.html',monte_count=400,risk_free = 0.03):
    """
    """
    path_plotly = self.path_dir_plotly_html + os.sep + file_name
    df = pd.read_excel(r'data\组合.xlsx',index_col=[0])
    returns = df.pct_change()
    returns.dropna(inplace=True)
    noa = 3

    # 蒙特卡洛随机模拟结果
    port_returns = []
    port_variance = []

    for p in range(monte_count):
        weights = np.random.random(noa)
        weights /= np.sum(weights)
        port_returns.append(np.sum(returns.mean() * 50 * weights)) # 加入模拟的均值
        port_variance.append(np.sqrt(np.dot(weights.T,
np.dot(returns.cov() * 50, weights)))) # 加入模拟的标准差

    port_returns = np.array(port_returns)

    port_variance = np.array(port_variance)
    color_array = (port_returns - risk_free) / port_variance # sharp 比,
不同的 sharp 比对应的颜色是不同的

```

将模拟结果按照 sharp 比进行排序并生成 colorbar

```

# 此处位置为 get_plotly_path_monte_markovitz 函数内部
trace1 = go.Scatter(
    x=port_variance,
    y=port_returns,
    mode='markers',
    marker=dict(
        size='6',
        color=color_array, # 通过一个可变的变量表示颜色，结果是绘图颜色可变
        colorscale='Viridis',
        # 设置 colorbar
        colorbar=dict(
            tickmode='linear',
            tick0=color_array.min(),
            dtick=(color_array.max() - color_array.min()) / 5,
        ),
        showscale=True,
    )
)
data = [trace1]

pyof.plot(data, filename=path_plotly, auto_open=False)
return path_plotly

```

本章主要介绍“图形用户界面”GUI。GUI 是图形用户界面（Graphical User Interface）的缩写，它是用户与计算机交互的窗口。本章将介绍 PyQt5 库，它是 Python 语言中用于开发 GUI 应用程序的库。本章将介绍 PyQt5 库的基本用法，包括如何创建窗口、添加控件、处理事件等。本章还将介绍 PyQt5 库的高级用法，包括如何创建自定义控件、如何实现多线程等。本章将介绍 PyQt5 库的常用控件，包括按钮、文本框、列表框等。本章将介绍 PyQt5 库的常用事件，包括单击、双击、拖拽等。本章将介绍 PyQt5 库的常用布局管理器，包括水平布局、垂直布局、网格布局等。本章将介绍 PyQt5 库的常用样式表，包括背景色、字体、边框等。本章将介绍 PyQt5 库的常用信号槽，包括信号发射、信号接收等。本章将介绍 PyQt5 库的常用动画，包括位置动画、颜色动画等。本章将介绍 PyQt5 库的常用窗口，包括主窗口、对话框、子窗口等。本章将介绍 PyQt5 库的常用工具，包括调试器、性能分析器等。本章将介绍 PyQt5 库的常用资源，包括文档、示例代码等。本章将介绍 PyQt5 库的常用问题，包括安装问题、运行问题、兼容性问题等。本章将介绍 PyQt5 库的常用技巧，包括快捷键、热键、拖拽等。本章将介绍 PyQt5 库的常用陷阱，包括内存泄漏、线程安全问题等。本章将介绍 PyQt5 库的常用最佳实践，包括代码组织、命名规范等。本章将介绍 PyQt5 库的常用扩展，包括第三方库、插件等。本章将介绍 PyQt5 库的常用未来展望，包括新功能、性能优化等。

## [11.4 PyQt 5 图形用户界面](#)

本章主要介绍“图形用户界面”GUI。GUI 是图形用户界面（Graphical User Interface）的缩写，它是用户与计算机交互的窗口。本章将介绍 PyQt5 库，它是 Python 语言中用于开发 GUI 应用程序的库。本章将介绍 PyQt5 库的基本用法，包括如何创建窗口、添加控件、处理事件等。本章还将介绍 PyQt5 库的高级用法，包括如何创建自定义控件、如何实现多线程等。本章将介绍 PyQt5 库的常用控件，包括按钮、文本框、列表框等。本章将介绍 PyQt5 库的常用事件，包括单击、双击、拖拽等。本章将介绍 PyQt5 库的常用布局管理器，包括水平布局、垂直布局、网格布局等。本章将介绍 PyQt5 库的常用样式表，包括背景色、字体、边框等。本章将介绍 PyQt5 库的常用信号槽，包括信号发射、信号接收等。本章将介绍 PyQt5 库的常用动画，包括位置动画、颜色动画等。本章将介绍 PyQt5 库的常用窗口，包括主窗口、对话框、子窗口等。本章将介绍 PyQt5 库的常用工具，包括调试器、性能分析器等。本章将介绍 PyQt5 库的常用资源，包括文档、示例代码等。本章将介绍 PyQt5 库的常用问题，包括安装问题、运行问题、兼容性问题等。本章将介绍 PyQt5 库的常用技巧，包括快捷键、热键、拖拽等。本章将介绍 PyQt5 库的常用陷阱，包括内存泄漏、线程安全问题等。本章将介绍 PyQt5 库的常用最佳实践，包括代码组织、命名规范等。本章将介绍 PyQt5 库的常用扩展，包括第三方库、插件等。本章将介绍 PyQt5 库的常用未来展望，包括新功能、性能优化等。

PyQtPyQtGUI PyQt GUI PyQt GUI

Python 3+PyQt 5 Python 2 Python 3 zwquant GUI zwquant Python 3 .py

zwquant 1 Chapter11/zwquant\_pyqt GUI

PyQt5/Chapter11/zwquant\_pyqt/zq902\_macd\_v2.py 11-12



```

        zwdr.my_pyqt_show(qx)
    else:
        zwdr.my_qunt_plot(qx)
zwdr.my_pyqt_show()
PyQt5/Chapter11/zwquant_pyqt/zwQTDraw.py
my_pyqt_show()
def my_pyqt_show(qx):
    from my_back_test_show import MainWindow
    from PyQt5.QtWidgets import QMainWindow, QApplication
    import sys
    app=QApplication(sys.argv)
    ui=MainWindow(qx)
    ui.showMaximized()
    # ui.show()
    sys.exit(app.exec_())
PyQt5/Chapter11/zwquant_pyqt/my_back_test_show.py

```

```

def __init__(self, qx=None, parent=None):
    """
    Constructor

    @param parent reference to the parent widget
    @type QWidget
    """
    super(MainWindow, self).__init__(parent)
    self.setupUi(self)
    if qx != None: # 与 zwquant 结合, qx 是 zwquant 的一个类实例
        self.qx = qx
        self.show_result(self.qx)
        self.matplotlibwidget_static.mpl.start_static_plot(self.qx)
    else: # 用于测试, 不需要 zwquant 也能运行, 方便快速开发自己的 GUI 界面
        self.show_result()

```

```

self.matplotlibwidget_static.mpl.start_static_plot()

```

在 `MainWindow` 类中调用 `zwquant` 的 `show_result()` 和 `matplotlibwidget_static.mpl.start_static_plot()` 方法

```

def show_result(self, qx=None):

    if qx != None: # 跑回测的话就传入回测数据
        list_result = qx.result_info
        pickle_file = open('my_list.pkl', 'wb') # 以 wb 方式写入
        pickle.dump(list_result, pickle_file) # 向 pickle_file 中写入
my_list
        pickle_file.close()
    else: # 不跑回测的话就读取测试数据
        pickle_file = open('my_list.pkl', 'rb') # 以 rb 方式读取
        list_result = pickle.load(pickle_file) # 读取以 pickle 方式写入的
文件 pickle_file
        pickle_file.close()

        list_result.append(['', '']) # 为了能够凑够 24*2 (原来是 23*2)
        len_index = 6
        len_col = 8
        list0, list1, list2, list3 = [list_result[6 * i:6 * i + 6] for i in
range(0, 4)]
        arr_result = np.concatenate([list0, list1, list2, list3], axis=1)
        self.tableWidget.setRowCount(len_index) # 设置行的数量
        self.tableWidget.setColumnCount(len_col) # 设置列的数量
        self.tableWidget.setHorizontalHeaderLabels(['回测内容', '回测结果'] *
4) # 设置垂直方向上的标题
        self.tableWidget.setVerticalHeaderLabels([str(i) for i in range(1,
len_index + 1)]) # 设置水平方向上的标题

        for index in range(len_index):
            for col in range(len_col):
                self.tableWidget.setItem(index, col,
QTableWidgetItem(arr_result[index, col]))
        self.tableWidget.resizeColumnsToContents() # 根据内容来调整列的宽度

```

show\_result() 调用 tableWidget 方法

11-13

PyQt5/Chapter11/zwquant\_pyqt/MatplotlibWidget.py

zwquant ax2=ax1.twinx() ax2

	回测内容	回测结果	回测内容	回测结果	回测内容	回测结果	回测内容	回测结果
1	交易总次数	13	平均日收益率	0.156	最长回测时间	182	开始时间	2015-01-01
2	交易总盈利	-8834.15	日收益率方差	0.0244	回测时间(最高点位)	2015-05-28	结束时间	2016-04-08
3	最终资产价值	\$13810.89	夏普比率	0.695, (0.05利率)	回测最高点位	15979.060	项目名称	maod20
4	最终现金资产价值	\$1165.85	无风险利率	0.05	回测最低点位	9937.720	策略名称	maod20
5	最终证券资产价值	\$12645.04	夏普比率(无风险)	0.824	时间周期	464 (Day)	策略参数变量 staVars[]	12 26 2015-01-01
6	累计回报率	38.11	最大回测率	0.0000	时间周期(交易日)	257 (Day)		

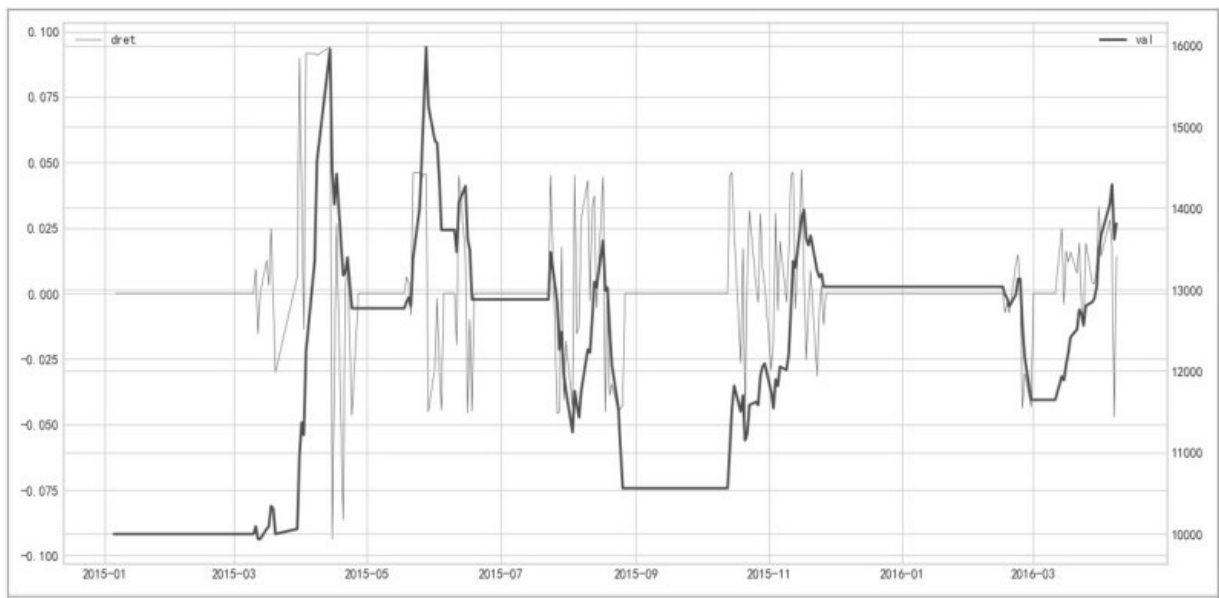
## □11-13

'''绘制静态图，可以在这里定义自己的绘图逻辑'''

```
def start_static_plot(self,qx=None):
    if qx != None: # 与 zwquant 结合, qx 是 zwquant 的一个类实例
        df = qx.qxLib.copy()
        df.set_index('date',inplace=True)
        df.rename_axis(lambda x: pd.to_datetime(x),inplace=True)
        ax1 = self.axes
        ax1.plot(df['dret'], color='green', label='dret', linewidth=0.5)
        ax1.legend(loc='upper left')
        ax2 = ax1.twinx()
        ax2.plot(df['val'], color='red', label='val', linewidth=2)
        ax2.legend(loc='upper right')
    else:# 用于测试, 不需要 zwquant 也能运行, 方便快捷开发自己的 GUI 界面
        t = arange(0.0, 3.0, 0.01)
        s = sin(2 * pi * t)
        self.axes.plot(t, s)
        self.axes.set_ylabel('静态图: Y 轴')
        self.axes.set_xlabel('静态图: X 轴')
        self.axes.grid(True)
```

## □□□□11-14□□□□





□11-14

```
PyQt5/Chapter11/zwquant_pyqt/tmp macd20_600401.csv macd20_qxLib.csv macd20_xtrdLib.csv
```

```

@pyqtSlot()
def on_pushButton_show_dataPre_clicked(self):
    """
    Slot documentation goes here.
    """
    if hasattr(self, 'qx'):# 与 zwquant 结合, 再进行下一步
        if hasattr(self.qx, 'path_dataPre'):
            os.system(np.random.choice(self.qx.path_dataPre)) # 随机选取
数据预处理的文件结果, 并打开

@pyqtSlot()
def on_pushButton_show_money_flow_clicked(self):
    """
    Slot documentation goes here.
    """
    if hasattr(self, 'qx'):# 与 zwquant 结合, 再进行下一步
        os.system(self.qx.fn_qxLib)

@pyqtSlot()
def on_pushButton_show_trade_flow_clicked(self):
    """
    Slot documentation goes here.
    """
    if hasattr(self, 'qx'):# 与 zwquant 结合, 再进行下一步
        os.system(self.qx.fn_xtrdLib)

```

11-15

pushButton_hide_output : QPushButton	
属性	值
> iconSize	16 x 16
> shortcut	
<b>checkable</b>	<input checked="" type="checkbox"/>
<b>checked</b>	<input checked="" type="checkbox"/>
autoRepeat	<input type="checkbox"/>
autoExclusive	<input type="checkbox"/>

图11-15

单击“Edit”→“配置连接”配置连接pushButton\_hide\_output和tableWidget  
 配置连接如图11-16所示  
 单击“OK”配置连接如图11-17所示

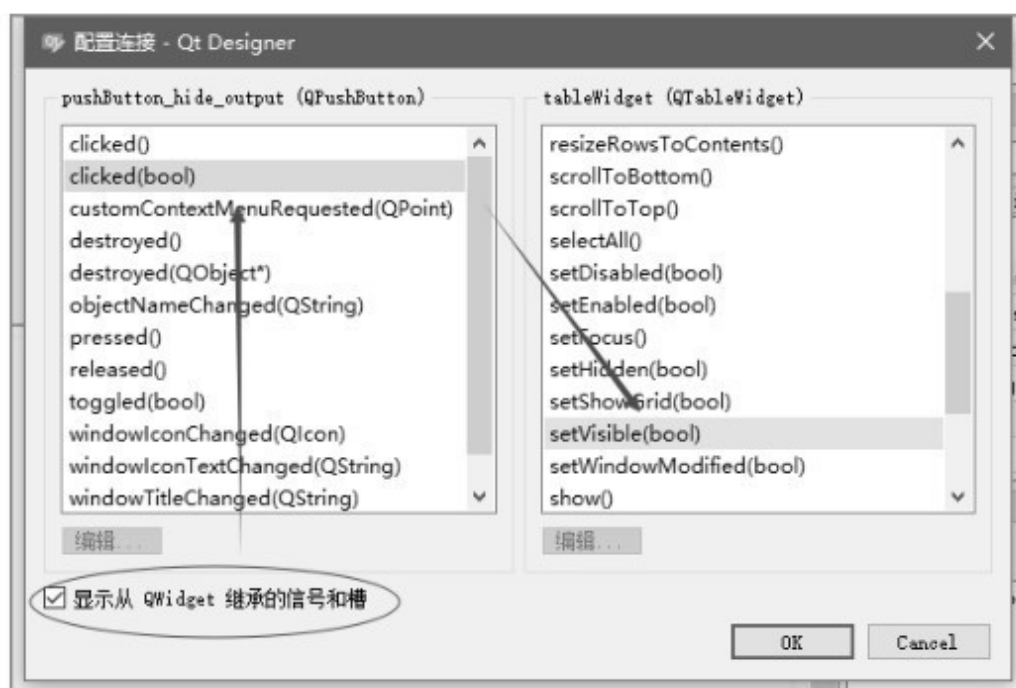
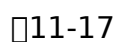


图11-16



```
self.pushButton_hide_output.clicked['bool'].connect(self
.tableWidget.setVisible)
```

PyQt 5.15.2 Qt 5.15.2  
PyQt 5.15.2 Qt 5.15.2

```
scrollArea  
    scrollArea  
  
scrollArea
```

## 11.5 PyQt 5

**Wind**

Wind

PyQt<http://www.cninfo.com.cn/cninfo-new/index>GUI

PyQt

### 11.5.1 背景

GUI GUI  
GUI  
PyQt5/Chapter11/juchao/craw.py

```
'''
模拟巨潮资讯的网络爬虫。
'''

import requests

def get_one_page_data(key, date_start='', date_end='', fulltext_str_
flag='false', page_num=1, pageSize=30,sortName='nothing',sortType='desc'):
    '''
    :param key: 搜索的关键词
    :param date_start:起始时间
    :param date_end: 终止时间
```

```

:param fulltext_str_flag:是否是内容搜索,默认为 false,即标题搜索
:param page_num: 要搜索的页码
:param pageSize: 每页显示的数量
:param sortName: 排序名称,对应关系为: '相关度': 'nothing', '时间':
'pubdate', '代码': 'stockcode_cat', 默认为相关度
:param sortType: 排序类型,对应关系为: '升序': 'asc', '降序': 'desc', 默
认为降序
:return: 总页码 和 当前页码的信息
'''
params = {'searchkey': key,
          'sdate': date_start,
          'edate': date_end,
          'isfulltext': fulltext_str_flag,
          'sortName': sortName,
          'sortType': sortType,
          'pageNum': str(page_num),
          'pageSize': str(pageSize)}
key_encode = requests.models.urlencode({'a': key}).split('=')[1]

url = 'http://www.cninfo.com.cn/cninfo-new/fulltextSearch/full'
headers = {'Accept': 'application/json, text/javascript, */*; q=0.01',
           'Accept-Encoding': 'gzip, deflate, sdch',
           'Accept-Language': 'zh-CN,zh;q=0.8',
           'Connection': 'keep-alive',
           'Cookie': 'JSESSIONID=7DF993E8D803E8672C6069F48399F60D;
cninfo_search_record_cookie=%s' % key_encode,
           'Host': 'www.cninfo.com.cn',
           'Referer': 'http://www.cninfo.com.cn/cninfo-new/fulltext
Search?code=&notautosubmit=&keyWord=%s' % key_encode,
           'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/51.0.2704.63 Safari/537.36
Qiyu/2.1.0.0', 'X-Requested-With': 'XMLHttpRequest'}
try:
    r = requests.get(url, headers=headers, params=params, timeout=20)
    # r.encoding = 'utf-8'
    page_content = r.json()
    page_value = page_content['announcements']
    total_page_num = page_content['totalpages']
    return total_page_num, page_value
except:
    return None, None

```





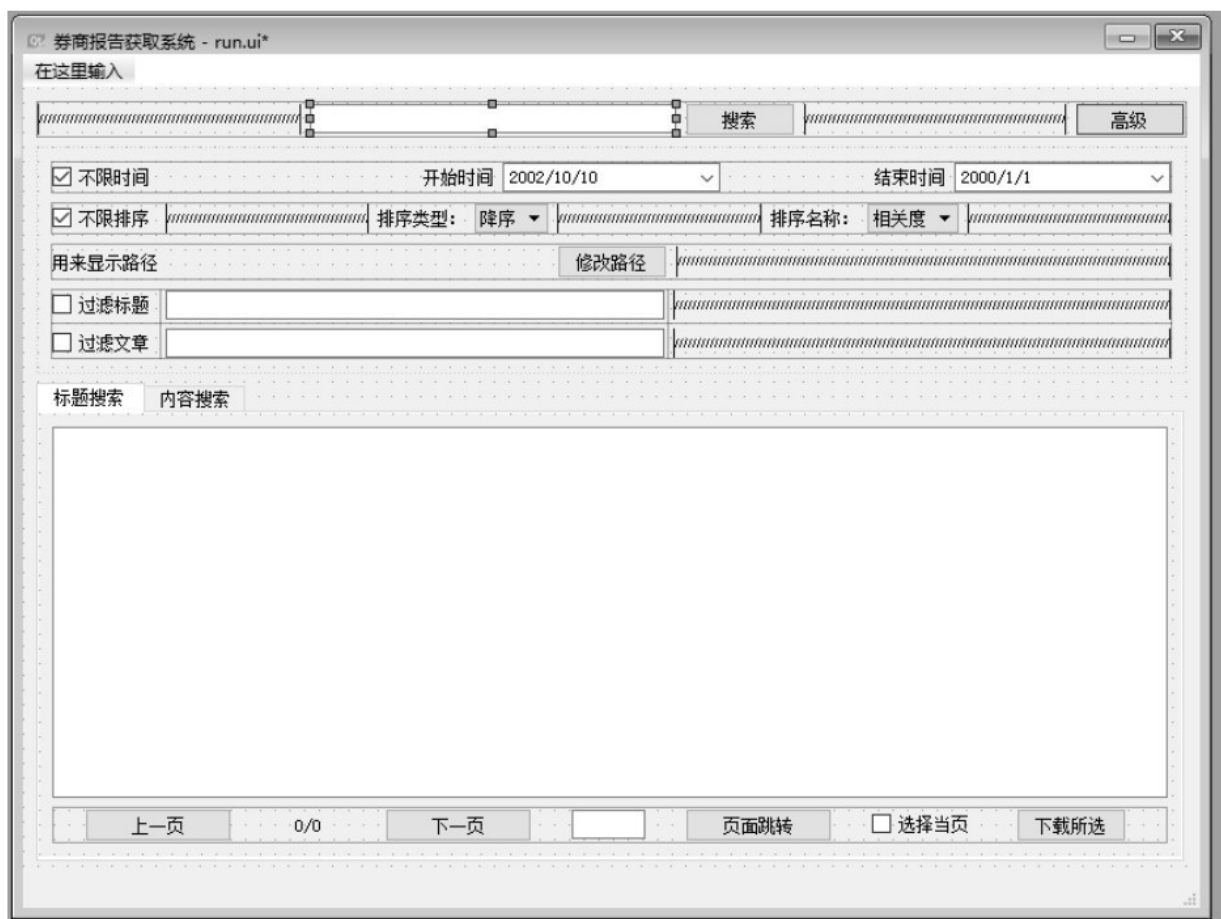


图11-18

## 2. 代码实现

```

def search(self):
    self.frame_advanced.hide()
    self.frame_advanced.show()
    self.frame_advanced.frame.show()
    self.frame_advanced.frame.show()

```

```

def __init__(self, parent=None):
    """
    Constructor

    @param parent reference to the parent widget
    @type QWidget
    """
    super(MainWindow, self).__init__(parent)
    self.setupUi(self)
    self.total_pages_content = 1
    self.total_pages_title = 1
    self.current_page_num_title = 1
    self.current_page_num_content = 1
    self.sort_type = 'desc'
    self.sort_name = 'nothing'
    self.comboBox_dict = {'相关度': 'nothing', '时间': 'pubdate', '代码':
'stockcode_cat', '升序': 'asc', '降序': 'desc'}
    self.frame_advanced.hide() # 默认隐藏 frame
    self.download_info_list = [] # 存储要下载的信息，每个元素都是字典形式的，
存储了要下载的标题、URL 等信息
    self.download_path = os.path.abspath(r'./下载')
    self.label_show_path.setText('当前保存目录为: ' + self.download_path)
    self.tableWidget_title_checked = Qt.Unchecked # 设置 tableWidget 的
默认选择方式
    self.tableWidget_content_checked = Qt.Unchecked
    self.select_title_page_info = set() # 记录 checkBox_select 选择的页面
信息
    self.select_content_page_info = set() # 记录 checkBox_select 选择的页面
信息
    self.filter_title_list = [] # 用来显示过滤标题的列表
    self.filter_content_list = [] # 用来显示过滤内容的列表

    '''下面四行代码一定要按照顺序执行，否则 self.start_time 与 self.end_time 这
两行代码会无效'''
    self.dateEdit.setDateTime(datetime.datetime.now())

```

```

self.dateEdit_2.setDateTime(datetime.datetime.now())
self.start_time = ''
self.end_time = ''

self.dateEdit.setEnabled(False)
self.dateEdit_2.setEnabled(False)
self.comboBox_type.setEnabled(False)
self.comboBox_name.setEnabled(False)
self.lineEdit_filter_content.setEnabled(False)
self.lineEdit_filter_title.setEnabled(False)

'''连接信号与槽'''
'显示或隐藏高级选项'
self.pushButton_setting_advanced.toggled['bool'].connect
(self.frame_advanced.setHidden)
'下载'
self.pushButton_download_select_title.clicked.connect
(self.download_pdf)
self.pushButton_download_select_content.clicked.connect
(self.download_pdf)
download_thread.signal.connect(self.show_status) # 子线程的信号连接
主线程的槽
'修改存储路径'
self.pushButton_change_save_path.clicked.connect
(self.change_save_path)
'tableWidget 相关'
self.tableWidget_title.itemChanged.connect(self.select_item)
self.tableWidget_content.itemChanged.connect(self.select_item)
self.tableWidget_title.cellClicked.connect(self.view_one_new)
self.tableWidget_content.cellClicked.connect(self.view_one_new)
'状态栏显示'
self.signal_status.connect(self.show_status) # 将状态栏信号绑定到槽
'在 lineEdit 控件上按下 Enter 键就可以触发搜索或跳转到页码'
self.lineEdit.returnPressed.connect(self.on_pushButton_search_
clicked)
self.lineEdit_filter_title.returnPressed.connect(self.on_
pushButton_search_clicked)
self.lineEdit_filter_content.returnPressed.connect
(self.on_pushButton_search_clicked)
self.lineEdit_content_page.returnPressed.connect
(self.pushButton_content_jump_to.click)
self.lineEdit_title_page.returnPressed.connect(lambda:

```

```

self.page_go('title_jump_to'))
    '页码跳转函数'

    self.pushButton_title_down.clicked.connect(lambda:
self.page_go('title_down'))
    self.pushButton_content_down.clicked.connect(lambda:
self.page_go('content_down'))
    self.pushButton_title_up.clicked.connect(lambda:
self.page_go('title_up'))
    self.pushButton_content_up.clicked.connect(lambda:
self.page_go('content_up'))
    self.pushButton_title_jump_to.clicked.connect(lambda:
self.page_go('title_jump_to'))
    self.pushButton_content_jump_to.clicked.connect(lambda:
self.page_go('content_jump_to'))
    '选择标题或内容'

    self.checkBox_select_title.clicked['bool'].connect
(self.select_checkBox)
    self.checkBox_select_content.clicked['bool'].connect
(self.select_checkBox)
    '显示/下载过滤操作'

    self.checkBox_filter_title.clicked['bool'].connect
(self.filter_enable)
    self.checkBox_filter_content.clicked['bool'].connect
(self.filter_enable)


    '初始化下载目录'

    if not os.path.isdir(self.download_path):
        os.mkdir(self.download_path)

```

### 3. 槽函数

槽函数是 Qt 中一个非常重要的概念，它允许你将一个信号与一个函数（槽）连接起来。当信号发生时，槽函数会被调用。槽函数的名称通常以 `on_` 开头，后面跟信号名称和 `_clicked`。例如，`on_pushButton_search_clicked`。

`@pyqtSlot()`

`def on_pushButton_search_clicked(self):`

"""

Slot documentation goes here.

"""

```

self.download_info_list=[]# 下载信息列表
self.current_page_num_title=1 # 当前页码标题1
self.current_page_num_content=1
self.update_tablewidget_title() # 更新标题
self.update_tablewidget_content() # 更新内容
# 当按下Enter键时，调用search函数
def __init__(self):
    self.lineEdit.returnPressed.connect(self.on_pushButton_search_clicked)
    self.lineEdit_filter_title.returnPressed.connect(
        self.on_pushButton_search_clicked)
    self.lineEdit_filter_content.returnPressed.connect(
        self.on_pushButton_search_clicked)
    # 初始化lineEdit和lineEdit_filter的文本
    self.lineEdit.setText('')
    self.lineEdit_filter_title.setText('')
    self.lineEdit_filter_content.setText('')
    self.update_tablewidget_title() # 更新标题
    self.update_tablewidget_content() # 更新内容
    update_tablewidget_title()
    update_tablewidget_content()

```

```

def update_tablewidget_title(self, page_num=1):
    '''更新tablewidget_title'''
    key_word = self.lineEdit.text()
    '''从网络爬虫中获取数据'''
    total_pages_title, dict_data_title = get_one_page_data(key_word,
fulltext_str_flag='false', page_num=page_num,
date_start=self.start_time, date_end=self.end_time,
sortName=self.sort_name, sortType=self.sort_type)
    '''把数据显示到表格上'''
    if total_pages_title != None:
        self.total_pages_title = total_pages_title
        self.show_tablewidget(dict_data_title, self.tableWidget_title,
clear_fore=False)
        self.label_page_info_title.setText('%d/%d' %
(self.current_page_num_title, self.total_pages_title)) # 更新当前页码信息

```

在update\_tablewidget\_title函数中，我们调用了show\_tablewidget函数，并传入了dict\_data\_title和self.tableWidget\_title两个参数。同时，我们还调用了self.label\_page\_info\_title.setText方法，将当前页码和总页码信息更新到界面上。

```

def show_tablewidget(self, dict_data, tableWidget, clear_fore=True):
    '''传入dict_data 与 tableWidget，以实现在tableWidget上面呈现dict_data'''
    '''提取自己需要的信息：'''
    if clear_fore == True: # 检测在搜索之前是否要清空下载购物车信息
        self.download_info_list = []

```

在show\_tablewidget函数中，我们首先检测了clear\_fore是否为True。如果是True，我们就清空了download\_info\_list列表。然后，我们提取了需要的信息。

最后，我们调用了self.show\_tablewidget函数，将提取出的信息展示在界面上。

# 调用show\_tablewidget函数



```
dict_target['flag']=flag
```

```
# 此处位置在函数 show_tablewidget 内部

'''从传入的网络爬虫抓取的数据中提取自己需要的数据'''
if len(dict_data) > 0:
    # key_word = self.lineEdit.text()
    len_index = len(dict_data)
    list_target = [] # 从 dict_data 中提取目标数据，基本元素是下面的
dict_target
    for index in range(len_index):
        dict_temp = dict_data[index]# 提取从服务器中返回的其中一行信息
        dict_target = {} # 从 dict_temp 中提取自己需要的信息，主要包括标题、内
容、时间、下载 URL 等
        '提取标题与内容'
        _temp_title = dict_temp['announcementTitle']
        _temp_content = dict_temp['announcementContent']
        for i in ['<em>', '</em>']: # <em>, </em>是服务器对搜索关键词添加的
标记，这里对它们剔除
            _temp_title = _temp_title.replace(i, '')
            _temp_content = str(_temp_content).replace(i, '')

        dict_target['title'] = _temp_title
        dict_target['content'] = _temp_content
```



```

        '提取时间'
        _temp = dict_temp['adjunctUrl']
        dict_target['time'] = _temp.split(r'/')[1]

        '提取 URL'
        id = _temp.split(r'/')[2].split('.')[0]
        download_url = 'http://www.cninfo.com.cn/cninfo-new/disclosure/
fulltext/download/{}?announceTime={}'.format(
            id, dict_target['time'])
        dict_target['download_url'] = download_url
        dict_target['flag'] = flag
        # print(download_url)
        '添加处理的结果'
        list_target.append(dict_target)

```

对提取出的数据进行过滤

```

# 此处位置在函数 show_tablewidget 内部

'''根据过滤规则进行自定义过滤，默认是不过滤的'''
df = DataFrame(list_target)
df = self.filter_df(df, filter_title_list=self.filter_
title_list, filter_content_list = self.filter_content_list)

'''过滤后，更新 list_target'''
_temp = df.to_dict('index')
list_target = list(_temp.values())

else: # '处理没有数据的情况'
    list_target = []

```

对提取出的数据进行过滤

对

```
# 此处位置在函数 show_tablewidget 内部
```

```
'''tableWidget 的初始化'''
```

```
list_col = ['time', 'title', 'download_url']
```

```
len_col = len(list_col)
```

```
len_index = len(list_target) # list_target 可能有所改变，需要重新计算长度
```

```
if tableWidget.objectName() == 'tableWidget_title':
```

```
    self.list_target_title = list_target
```

```
else:
```

```
    self.list_target_content = list_target
```

```
tableWidget.setRowCount(len_index) # 设置行数
```

```
tableWidget.setColumnCount(len_col) # 设置列数
```

```
tableWidget.setHorizontalHeaderLabels(['时间', '标题', '查看']) # 设置垂直方向上的名称
```

```
tableWidget.setVerticalHeaderLabels([str(i) for i in range(1, len_index + 1)]) # 设置水平方向上的名称
```

```
tableWidget.setCornerButtonEnabled(True) # 点击左上角进行全选
```

```
□□□□□□□□□□□□□□□□
```

```

# 此处位置在函数 show_tablewidget 内部

'''填充 tableWidget 的数据'''
for index in range(len_index):
    for col in range(len_col):
        name_col = list_col[col]
        if name_col == 'download_url':
            item = QTableWidgetItem('查看')
            item.setTextAlignment(Qt.AlignCenter)
            font = QFont()
            font.setBold(True)
            font.setWeight(75)
            item.setFont(font)
            item.setBackground(QColor(218, 218, 218))
            item.setFlags(Qt.ItemIsUserCheckable | Qt.ItemIsEnabled)
            tableWidget.setItem(index, col, item)
        elif name_col == 'time':
            item = QTableWidgetItem(list_target[index][name_col])
            item.setFlags(Qt.ItemIsUserCheckable |
                           Qt.ItemIsEnabled)
            '''查看当前行的内容是否已经在下载购物车中，如果在就设置为选中'''
            if list_target[index] in self.download_info_list:
                item.setCheckState(Qt.Checked)
            else:
                item.setCheckState(Qt.Unchecked)
            tableWidget.setItem(index, col, item)
        else:
            tableWidget.setItem(index, col,
QTableWidgetItem(list_target[index][name_col]))
    # tableWidget.resizeColumnsToContents()
    tableWidget.setColumnWidth(1, 500)

```

11-19



```

        item.setBackground(QColor(218,218,218))
        item.setFlags(Qt.ItemIsUserCheckable |
Qt.ItemIsEnabled)
        tableWidget.setItem(index,col,item)
        #2#### QTableWidgetItem#### check ##### CheckBox #
#####check#####
        #3#####
#####
        item=QTableWidgetItem(list_target[index][name_col])
        item.setFlags(Qt.ItemIsUserCheckable |
Qt.ItemIsEnabled)
        ""#####""
        if list_target[index]in self.download_info_list:
            item.setCheckState(Qt.Checked)
        else:
            item.setCheckState(Qt.Unchecked)
        tableWidget.setItem(index,col,item)
        # 4 # # # # # # # # # # 500
tableWidget.setColumnWidth(1,500)#####
tableWidget.resizeColumnsToContents()#
# tableWidget.resizeColumnsToContents()
tableWidget.setColumnWidth(1,500)
4.####
#####11-20#####

```

```

def __init__(self):
    self.pushButton_title_down.clicked.connect(lambda:
self.page_go('title_down'))
    self.pushButton_content_down.clicked.connect(lambda:
self.page_go('content_down'))
    self.pushButton_title_up.clicked.connect(lambda:
self.page_go('title_up'))
    self.pushButton_content_up.clicked.connect(lambda:
self.page_go('content_up'))
    self.pushButton_title_jump_to.clicked.connect(lambda:
self.page_go('title_jump_to'))
    self.pushButton_content_jump_to.clicked.connect(lambda:
self.page_go('content_jump_to'))

def page_go(self, page):
    """
    """
    self.lineEdit_content_page.returnPressed.connect(self
.pushButton_content_jump_to.click)
    self.lineEdit_title_page.returnPressed.connect(lambda:
self.page_go('title_jump_to'))

```

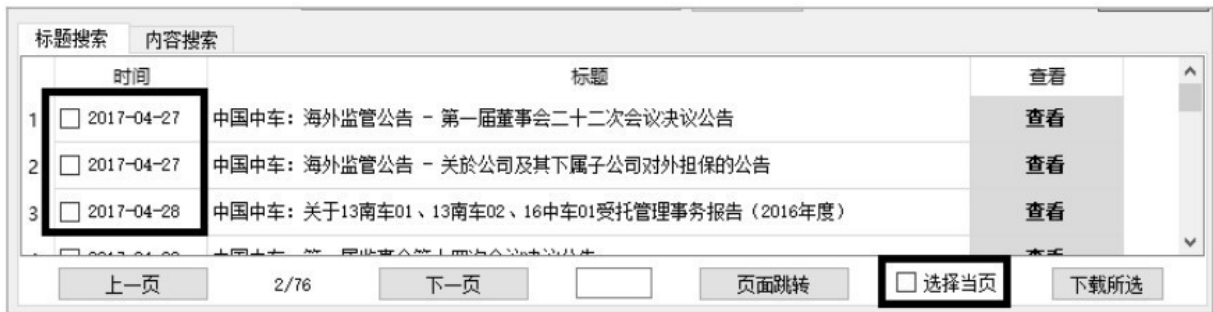
def page\_go(self, go\_type):  
'''页面跳转主函数'''  
if go\_type == 'title\_down': # 触发“下一页”按钮  
\_temp = self.current\_page\_num\_title  
self.current\_page\_num\_title += 1  
if 1 <= self.current\_page\_num\_title <= self.total\_pages\_title: #  
如果待跳转的页面真实、有效，则继续；否则不进行跳转  
self.update\_tablewidget\_title(page\_num=self.current\_  
page\_num\_title)  
else:  
self.current\_page\_num\_title = \_temp

def page\_go(self, go\_type):  
'''页面跳转主函数'''  
if go\_type == 'title\_down': # 触发“下一页”按钮  
\_temp = self.current\_page\_num\_title  
self.current\_page\_num\_title += 1  
if 1 <= self.current\_page\_num\_title <= self.total\_pages\_title: #  
如果待跳转的页面真实、有效，则继续；否则不进行跳转  
self.update\_tablewidget\_title(page\_num=self.current\_  
page\_num\_title)  
else:  
self.current\_page\_num\_title = \_temp  
  
if go\_type == 'title\_jump\_to':  
\_temp = self.current\_page\_num\_title  
self.current\_page\_num\_title = int(self.lineEdit\_title\_p  
age.text())  
if 1 <= self.current\_page\_num\_title <= self.total\_pages\_title:  
self.update\_tablewidget\_title(page\_num=self.current\_  
page\_num\_title)

```
self.current_page_num_title=_temp
```

## 5.□□□□

11-21



□11-21

tableWidget
checkBox

'tableWidget[]'
-----------------

```
self.tableWidget_title.itemChanged.connect(self.select_item)
```

```
self.tableWidget_content.itemChanged.connect(self.select item)
```

'□□□□□□□□'

```
self.checkBox_select_title.clicked['bool']
connect(self.select checkBox)
```

```
self.checkBox_select_content.clicked['bool'].  
connect(self.select checkBox)
```

```
select item
```



```

def select_item(self, item):
    '''处理选择 item 的主函数'''
    # print('item+change')
    column = item.column()
    row = item.row()
    if column == 0: # 只针对第一列
        if item.checkState() == Qt.Checked:
            if item.tableWidget().objectName() == 'tableWidget_title':
                download_one = self.list_target_title[row]
            else:
                download_one = self.list_target_content[row]
            if download_one not in self.download_info_list:
                self.download_info_list.append(download_one)
                self.signal_status.emit('select_status', [])
        else:
            if item.tableWidget().objectName() == 'tableWidget_title':
                download_one = self.list_target_title[row]
            else:
                download_one = self.list_target_content[row]
            if download_one in self.download_info_list:
                self.download_info_list.remove(download_one)
                self.signal_status.emit('select_status', [])

```

当column=0时，item.checkState 为Qt.Checked  
 则self.download\_info\_list添加download\_one  
 并emit select\_status信号，参数为[]  
 当item.checkState为Qt.Unchecked时  
 self.download\_info\_list删除download\_one  
 并emit select\_status信号

self.signal\_status.emit('select\_status',[]) 为emit信号  
 参数为字符串  
 “select\_status”

```

def select_checkBox(self, bool):
    sender=self.sender() # sender()为发出信号的对象
    if sender.objectName()=='checkBox_select_title':

```

```

        self.select_checkBox_one(sender,self.tableWidget_title)
    elif
sender.setObjectName()=='checkBox_select_content':
        self.select_checkBox_one(sender,self.tableWidget_content)
    """

```

```

def select_checkBox_one(self, sender, tableWidget):
    if sender.checkState() == Qt.Checked:
        self.select_tableWidget(tableWidget)
        if tableWidget.setObjectName() == 'tableWidget_title':
            self.select_title_page_info.add(self.current_page_num_title)
        elif tableWidget.setObjectName() == 'tableWidget_content':
            self.select_content_page_info.add(self.current_page_num_content)
        else:
            self.select_tableWidget_clear(tableWidget)
            if tableWidget.setObjectName() == 'tableWidget_title':
                if self.current_page_num_title in self.select_title_page_info:
                    self.select_title_page_info.remove(self.current_page_num_title)
            elif tableWidget.setObjectName() == 'tableWidget_content':
                if self.current_page_num_content in self.select_content_page_info:
                    self.select_content_page_info.remove(self.current_page_num_content)

```

```

    """
    self.select_tableWidget(tableWidget)"""
    self.select_tableWidget_clear(tableWidget)"""
    """
    """
    """show_tablewidget"""

```

```
def select_tableWidget(self,tableWidget):
```

```
    '''清空tableWidget'''
```

```
    row_count=tableWidget.rowCount()
```

```
    for index in range(row_count):
```

```
        item=tableWidget.item(index,0)
```

```
        if item.checkState()==Qt.Unchecked:
```

```
            item.setCheckState(Qt.Checked)
```

```
def select_tableWidget_clear(self,tableWidget):
```

```
    '''清空tableWidget'''
```

```
    row_count=tableWidget.rowCount()
```

```
    for index in range(row_count):
```

```
        item=tableWidget.item(index,0)
```

```
        if item.checkState()==Qt.Checked:
```

```
            item.setCheckState(Qt.Unchecked)
```

## 6. 测试

在main函数中，我们使用以下代码来测试我们的程序。

我们使用以下代码来测试我们的程序。

测试代码

- PyQt 测试

- PyQt 测试

Python 测试

PyQt 测试 Python threading 测试

PyQt QThread 测试

```
def __init__(self):
```

```
    '''
```

```
    self.pushButton_download_select_title.clicked.
```

```

connect(self.download_pdf)
    self.pushButton_download_select_content.clicked.
connect(self.download_pdf)
    download_thread.signal.connect(self.show_status) #
#####
download_thread = WorkThread()#####
WorkThread()#####
download_thread.signal.connect(self.show_status)#####
#####signal#####show_status#####
#####self.download_pdf#####
def download_pdf(self):
    '''PDF'''
    if download_thread.isRunning()==True:
        QMessageBox.warning(self,'!', '#####
        ',QMessageBox.Yes)
        return None
    download_thread.download_list=self.download_info_li
st.copy()
    download_thread.download_path=copy.copy(self.dow
nload_path)
    download_thread.start()
    #####“”
    #####
    #####download_thread#####
    #####download_threaddownload_thread.start()
    ##### WorkThread ##### WorkThread #####
threading.Thread#####

```

```
class WorkThread(QThread):
    #声明一个包括 str 和 list 类型参数的信号
    signal = pyqtSignal(str, list)

    def __init__(self):
        self.download_list = self.download_path = []
        self.download_list_err = []
        self.filter_content_list = self.filter_title_list = []
        super(WorkThread, self).__init__()

    def main_download(self, download_list, download_path,
download_status='download_status'):
        count_all = len(download_list)
        count_err = count_right = count_num = 0
```

```

self.download_list_err = []
for key_dict in download_list:
    count_num += 1
    download_url = key_dict['download_url']
    time = key_dict['time']
    title = key_dict['title']
    total_title = time + '_' + title
    total_title = total_title.replace(':', ': ')
    total_title = total_title.replace('?', '? ')
    total_title = total_title.replace('*', '★')

    file_path = download_path + os.sep + '%s.pdf' % total_title
    if os.path.isfile(file_path) == True: # 若文件已经存在, 则默认为
下载成功

        count_right += 1
        signal_list = [count_num, count_all, count_right,
count_err, title]
        self.signal.emit(download_status, signal_list) # 循环结束
后发出信号

        continue
    else:
        f = open(file_path, "wb") # 先建立一个文件, 以免其他线程重复
建立这个文件

        try:
            r = requests.get(download_url, stream=True)
            data = r.raw.read()
        except:
            self.download_list_err.append(key_dict)
            count_err += 1
            f.close()
            os.remove(file_path) # 文件下载失败, 要先关闭 open 函数,
然后删除文件

            signal_list = [count_num, count_all, count_right,
count_err, title]
            self.signal.emit(download_status, signal_list) # 循环
结束后发出信号

            continue
        f.write(data)
        f.close()
        count_right += 1
        signal_list = [count_num, count_all, count_right,
count_err, title]

```



```

def show_status(self, type, list_args):
    if type == 'download_status':
        count_num, count_all, count_right, count_err, title = list_args
        self.statusBar().showMessage(
            '完成:{0}/{3}, 正确:{1}, 错误: {2}, 本次下载:
{4}'.format(count_num, count_right, count_err, count_all, title))
    if type == 'download_status_err':
        count_num, count_all, count_right, count_err, title = list_args
        self.statusBar().showMessage(
            '重新下载失败: 完成:{0}/{3}, 正确:{1}, 错误: {2}, 本次下载:
{4}'.format(count_num, count_right, count_err, count_all, title))
    if type == 'select_status':
        self.statusBar().showMessage('已选择: %d' %
len(self.download_info_list))
    if type == 'change_save_path_status':

        self.statusBar().showMessage('保存目录修改为: %s' %
self.download_path)
    if type == 'clear':
        self.statusBar().showMessage(' ')

```

3. 在下载过程中，如果发生错误，会弹出一个对话框，提示用户是否重新下载。如果用户选择“是”，则会重新下载。如果用户选择“否”，则会退出程序。

```

def run(self):
    self.main_download(self.download_list,self.download_path,
download_status='download_status')
    self.main_download(self.download_list_err,self.download_path,download_status='download_status_err')
    self.main_download(self.download_list_err,self.download_path,download_status='download_status_err')

```

7. 运行



11-22 “”PDF

标题搜索		内容搜索	
	时间	标题	查看
1	<input type="checkbox"/> 2017-05-22	中国中车：海外监管公告 - 第一届董事会第二十三次会议决议公告	查看
2	<input type="checkbox"/> 2017-05-23	中国中车：第一届董事会第二十三次会议决议公告	查看
3	<input type="checkbox"/> 2017-05-23	中国中车：第一届董事会第二十三次会议决议公告	查看
4	<input type="checkbox"/> 2017-05-23	中国中车：独立董事关于有关事项的独立意见	查看
5	<input type="checkbox"/> 2017-05-18	中国中车：2016年公司债券（第一期）跟踪评级报告（2017）	查看

11-22

```
__init__
self.tableWidget_title.cellClicked.connect(self.view_one_new)
self.tableWidget_content.cellClicked.connect(self.view_one_new)
def view_one_new(self,row,column):
    ""
    sender=self.sender()
    if column==2: # 
        if sender.objectName()=='tableWidget_title':
            download_one=self.list_target_title[row]
        else:
            download_one=self.list_target_content[row]
            download_path=copy.copy(self.download_path)
            view_thread=threading.Thread(target=self.view_one_new_thread,args=(download_path,download_one),daemon=True)
```

view\_thread.start()

PDF  
QThread  
threading

```
def view_one_new_thread(self, download_path, download_one):  
    '''查看功能的多线程程序'''  
    download_url = download_one['download_url']  
    title = download_one['title']  
    title = title.replace(':', ': ')  
    title = title.replace('?', '? ')  
    title = title.replace('*', '★')  
  
    path = download_path + os.sep + '%s.pdf' % title  
    if not os.path.isfile(path):  
        try:  
            r = requests.get(download_url, stream=True)  
            data = r.raw.read()  
        except:  
            return  
        f = open(path, "wb")  
        f.write(data)  
        f.close()  
        os.system(path)
```

PDF  
PDF  
PDF  
os.system(path)

## 8. 日期

11-23

☒ 不限时间
 开始时间 
 结束时间

☒ 不限排序
 排序类型: 
 排序名称:

当前保存目录为: D:\zw\_own\PyQt\my\_pyqt\_book\Chapter10\juchao\下载

## 11-23

“”

```
@pyqtSlot(bool)
def on_checkBox_sort_flag_clicked(self, checked):
    if checked == True: # 恢复默认的排序
        self.comboBox_name.setEnabled(False)
        self.comboBox_type.setEnabled(False)
        self.sort_name = 'nothing'
        self.sort_type = 'desc'
    elif self.comboBox_name.currentText() == '相关度': # 对于相关度,
        有些特殊
        self.comboBox_name.setEnabled(True)
        self.comboBox_type.setEnabled(False) # 上面
        comboBox_name.currentText()=="相关度", 则这个控件不可用。这是模拟官网的操作
        self.sort_name = 'nothing'
        self.sort_type = 'desc'
    else: # 对于其他的, 则设置对应的参数
        self.comboBox_name.setEnabled(True)
        self.comboBox_type.setEnabled(True)
        sort_name = self.comboBox_name.currentText()
        sort_type = self.comboBox_type.currentText()

        self.sort_name = self.comboBox_dict[sort_name]
        self.sort_type = self.comboBox_dict[sort_type]
```

sort\_name sort\_type

“”

“”

comboBox

```

@pyqtSlot(str)
def on_comboBox_name_currentTextChanged(self, p0):
    if p0 == '相关度':
        self.comboBox_name.setEnabled(True)
        self.comboBox_type.setEnabled(False)
        self.sort_name = 'nothing'
        self.sort_type = 'desc'
    else:
        self.comboBox_name.setEnabled(True)
        self.comboBox_type.setEnabled(True)

        sort_name = self.comboBox_name.currentText()
        self.sort_name = self.comboBox_dict[sort_name]

@pyqtSlot(str)
def on_comboBox_type_currentTextChanged(self, p0):
    sort_type = self.comboBox_type.currentText()
    self.sort_type = self.comboBox_dict[sort_type]

```

## 9. 日期选择

日期选择功能实现

```

@pyqtSlot(QDate)
def on_dateEdit_dateChanged(self, date):
    self.start_time = self.get_dateEdit_time(self.dateEdit)

@pyqtSlot(QDate)
def on_dateEdit_2_dateChanged(self, date):
    self.end_time = self.get_dateEdit_time(self.dateEdit_2)

```

## 10. 数据导出

数据导出功能实现

数据导出功能实现 GUI 界面设计 11-24



```

def filter_enable(self, bool):
    sender=self.sender()
    if sender.objectName()=='checkBox_filter_title':
        if bool==True:
            self.lineEdit_filter_title.setEnabled(True)
        else:
            self.lineEdit_filter_title.setEnabled(False)
    elif sender.objectName()=='checkBox_filter_content':
        if bool==True:
            self.lineEdit_filter_content.setEnabled(True)
        else:
            self.lineEdit_filter_content.setEnabled(False)
    filter_enable 0000000000000000"00000" checkBox 00——000
    "0000"checkBox00000lineEdit000000000000000000000000000000
    0000show_tablewidget0000000000000000
    ""000000000000""
    if self.lineEdit_filter_title.isEnabled()==True:
        filter_text=self.lineEdit_filter_title.text()
        self.filter_title_list=self.get_filter_list(filter_text)
    else:
        self.filter_title_list=[]
    if self.lineEdit_filter_content.isEnabled()==True:
        filter_text=self.lineEdit_filter_content.text()
        self.filter_content_list=self.get_filter_list(filter_text)
    else:
        self.filter_content_list=[]

```

```

        self.lineEdit_filter_title.setText(self.lineEdit_filter_content.text())
        self.lineEdit_filter_title.setText(self.lineEdit_filter_content.text())
        self.lineEdit_filter_title.setText(self.lineEdit_filter_content.text())

    def get_filter_list(self, filter_text):
        filter_text = re.sub(r'[\s()&]', '', filter_text) # 去除空格、括号、&
        filter_list = filter_text.split('&')
        return filter_list

    def filter_df(self, df, filter_title_list, filter_content_list):
        """根据filter_title_list和filter_content_list对df进行过滤"""
        df = DataFrame(list_target)
        df = self.filter_df(df, filter_title_list=self.filter_title_list, filter_content_list=self.filter_content_list)
        _temp = df.to_dict('index')
        list_target = list(_temp.values())
        df = DataFrame(list_target)
        df = self.filter_df(df, filter_title_list=self.filter_title_list, filter_content_list=self.filter_content_list)

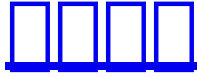
```





PyQt

PyQt



[1]PyQt 4 5 6 7 8  
[1].[http://pyqt.sourceforge.net/Docs/PyQt5/class\\_reference.html](http://pyqt.sourceforge.net/Docs/PyQt5/class_reference.html).

[2]Qt 5.15.2.<http://doc.qt.io/qt-5/index.html>.

[3]Qt 5.15.2.<http://www.kuqin.com/qtdocument/>.

[4]Python 3.7.4. Automate the Boring Stuff with Python.<https://automatetheboringstuff.com/#toc>.

[5]Python3 3.4.0.  
[5].<https://docs.python.org/3.4/index.html#>.

[6]Python2 2.7.18.  
[6].<https://docs.python.org/2/library/index.html>.

[7]Python3 3.7.4.<http://python.usyiyi.cn/>.